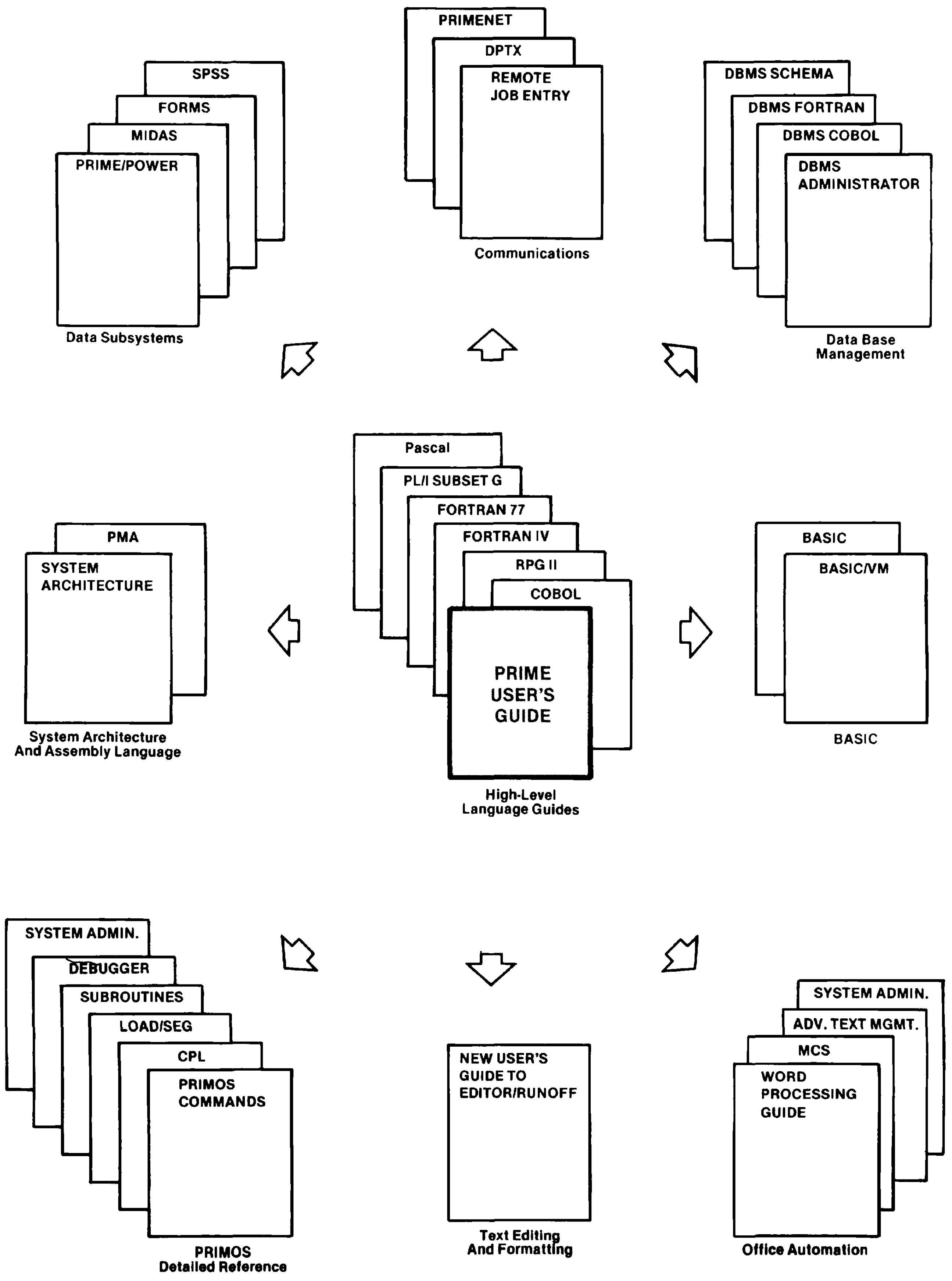
THE PRIME USER'S GUIDE

PR1ME Computer

PDR 4130



SPSS
FORMS
MIDAS
PRIME/POWER
Data Subsystems
PRIMENET
DPTX
REMOTE JOB ENTRY
Communications
DBMS SCHEMA
DBMS FORTRAN
DBMS COBOL
DBMS ADMINISTRATOR
Data Base Management
PMA
SYSTEM ARCHITECTURE
System Architecture And Assembly Language
Pascal
PL/I SUBSET G
FORTRAN 77
FORTRAN IV
RPG II
COBOL
PRIME USER'S GUIDE
High-Level Language Guides
BASIC
BASIC/VM
BASIC
SYSTEM ADMIN.
DEBUGGER
SUBROUTINES
LOAD/SEG
CPL
PRIMOS COMMANDS
PRIMOS Detailed Reference
NEW USER'S GUIDE TO EDITOR/RUNOFF
Text Editing And Formatting
SYSTEM ADMIN.
ADV. TEXT MGMT.
MCS
WORD PROCESSING GUIDE
Office Automation

The Prime User's Guide PDR 4130

This guide documents the software operation of the Prime Computer and its supporting systems and utilities as implemented at Master Disk Revision Level 18 (Rev. 18).

PR1ME Computer, Inc.
5ØØ Old Connecticut Path
Framingham, Massachusetts Ø17Ø1

ACKNOWLEDGEMENTS

We wish to thank the members of the documentation team and also the non-team members, both customer and Prime, who contributed to and reviewed this book.

First Printing December 1980

All correspondence on suggested changes to this document should be directed to:

Alice Landy
Technical Publications Department
Prime Computer, Inc.
500 Old Connecticut Path
Framingham, Massachusetts 01701

CONTENTS

Part I
Using Prime Documentation

SECTION 1

INTRODUCTION

WHAT THIS BOOK CONTAINS

The Prime User's Guide is an introduction and overview to programming in a high-level language on a Prime computer. It contains all the information new users need to get started on a Prime system, and provides a road map for new and experienced users alike that tells what's available for Prime computers and where to locate information about it.

This guide is divided into four parts.

Part I contains an introduction (Section 1), which tells how to use this book and provides an annotated guide to Prime's features and documentation.

Part II introduces users to PRIMOS (Prime's Operating System) and carries them step by step through the acts of creating and running a program, as follows:

- Section 2 introduces Prime's operating system, PRIMOS, and its file management system (FMS).
- Section 3 tells how to access the system: how to log in; how to create, manipulate, list and delete files and directories; and how to log out when you're done.
- Section 4 explains how to enter files (programs, text files, and data files), using Prime's editor; and how to get files printed on the line printer.
- Section 5 provides an introduction to compiling programs under PRIMOS. Simple programs can be compiled from the information given in this guide. For more complex programs, or programs for which the programmer wishes to use the advanced features of Prime's compilers, the programmer should consult the specific language reference guide.
- Section 6 provides an introduction to linking and loading programs with Prime's two loaders, SEG and LOAD. The information in this section enables users to load simple programs. The language guides provide information on language-specific features; the LOAD and SEG Reference Guide provides full information on advanced techniques.
- Section 7 provides an introduction to executing programs interactively. (Language-specific details on execution and debugging are provided by your language guide.)

- Section 8 introduces Prime's command procedure language, CPL, and shows how to write command procedure files for interactive or non-interactive running of programs.

- Section 9 tells how to create command input and output files for the non-interactive running of programs, how to execute command files from the terminal, and how to execute command files as phantoms (i.e., as independent processes not connected with your terminal).

- Section 10 provides full information on how to execute programs using Prime's batch processing environment.

Part III, System Facilities, provides an introduction to the resources available on your Prime system.

- Section 11 tells how to use four file-handling utilities:

  - SORT, which sorts and merges files

  - CMPF, which compares files and notes disparities

  - MRGF, which creates one updated file out of several disparate files

  - FUTIL, which moves, copies, lists, and deletes both files and complete directories

- Section 12 explains how to handle magnetic tapes, punched cards, and punched paper tapes on Prime.

- Section 13 explains PRIMENET, Prime's networking facility, and tells how users can take advantage of it.

- Section 14 provides a selected list of important subroutines and libraries available for use by high-level language programs.

Part IV provides a more advanced look at PRIMOS. In particular, it duscusses several ways in which you can alter the command environment on a terminal-by-terminal or program-by-program basis.

- Section 15 shows how you can define your own abbreviations for PRIMOS commands (via the ABBREV command) and how you can modify the system prompts with the RDY command.

- Section 16 explains PRIMOS's condition mechanism and shows how users can write their own on-units (error-handling subroutines).

In addition to the body of the text, this guide provides the following appendices:

- A glossary of terms used in Prime documentation

- A list of system defaults and constants

- The ASCII character set
- A list of system error messages
- A summary of Editor commands

HOW TO USE THIS BOOK

We suggest that you:

- Read Sections 1-4 before beginning to work on the system.
- Read Sections 6-10 before you try to compile, load or run programs.
- Use Sections 11 through 14 as reference sections:
  - Section 11 if you need to sort, compare, or merge files, or move whole directories from place to place
  - Section 12 if you need to use mag tapes, cards, or paper tape
  - Section 13 if the computer you work on is part of a network
  - Section 14 to find out whether PRIMOS has a subroutine or utility that does some task you need to do, or whether you'll have to write your own
- Read Sections 15 and 16 when you've become somewhat familiar with the system, to discover some more sophisticated conveniences PRIMOS can offer you.
- Refer to the glossary in Appendix A if you encounter any terms you don't recognize.

HOW TO USE THE REST OF PRIME'S DOCUMENTATION

If this User's Guide provided <u>all</u> the information you'd ever need to do anything, it would be about a foot thick. Therefore, Sections 2 through 16 contain enough information to get you started on just about everything. And the rest of this section supplies a road map to all our other documentation: the books that do tell you "all you need to know." (Titles followed by asterisks document separately priced products.)

The Central Guides

The relationship between these books is illustrated in Figure 1-1. This user's guide is the center: the starting place. Backing it up are the high-level language guides, which:

- Provide full language reference materials
- Explain the compilers in detail, showing the use of all options
- Explain any language-specific techniques of program development
- Discuss advanced techniques for loading, optimizing, and debugging programs

Language guides currently available are:

- The FORTRAN 77 Reference Guide*
- The FORTRAN Reference Guide
- The COBOL Reference Guide*
- The PL/I Subset G Reference Guide*
- The RPG II Reference Guide (and the RPG II Debugging Template)
- The Pascal Reference Guide*

More Detailed References

The commands and utilities explained in this guide and the language guides will carry most applications programmers through most of their work. For those who need more detailed references, each topic discussed in this book is treated more fully in our reference guides. The reference guides that applications programmers are most likely to use are:

- The PRIMOS Commands Reference Guide, which discusses all PRIMOS level commands available to the user.
- The CPL User's Guide, which describes how to use Prime's Command Procedure Language.
- The Subroutines Reference Guide, which tells how to incorporate into your own programs the various subroutines supplied by Prime.
- The LOAD and SEG Reference Guide, which provides a full discussion of Prime's loaders for users interested in taking advantage of their advanced features.

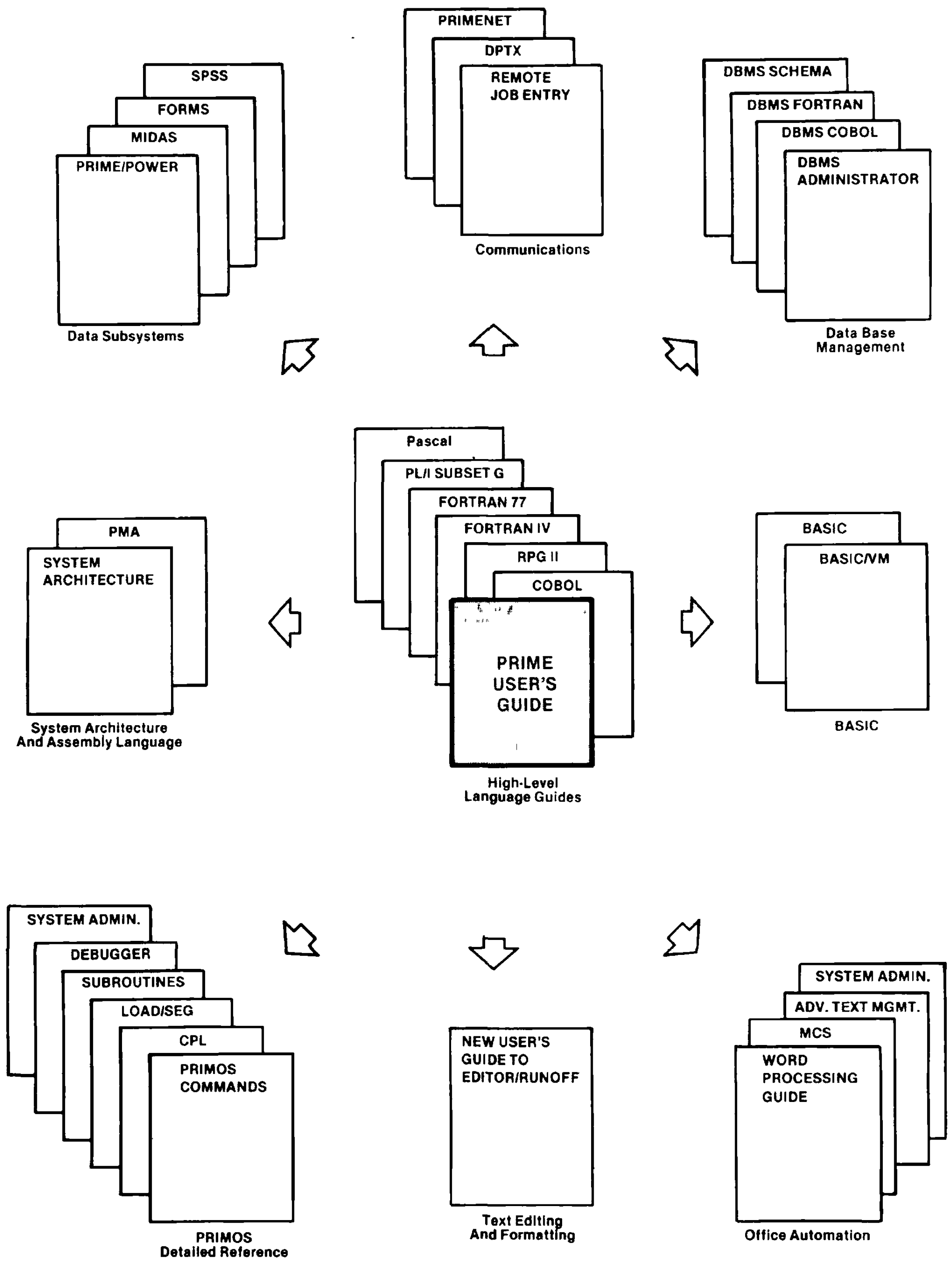


Figure 1-1. Organization of Prime Documentation

- The Source-Level Debugger Reference Guide,* which provides both introductory and full discussions on the use of Prime's interactive debugger for FORTRAN, FORTRAN 77, and PL/I programs.

A reference guide for operators and system administrators is:

- The System Administrator's Guide, which tells how to configure, bring up, and maintain a Prime system.

BASIC

BASIC is implemented on Prime computers as a fully interactive, self-contained environment. Working in BASIC, a programmer can write, compile, execute, and debug a program while remaining inside the BASIC environment. Prime's guides to working with BASIC, therefore, are similarly self-contained, providing both full explanations of all BASIC features and all introductory material needed to get the new user onto the system. The guides are:

- Interpretive BASIC

- The BASIC/VM Programmer's Guide*

Assembly Language

For assembly language programmers, and for anyone interested in learning about Prime's computer architecture, there are:

- The PMA Programmer's Guide

- The System Architecture Reference Guide

Prime also supplies a number of guides that deal with more specific applications.

Text Editing

For users concerned with text editing or formatted printouts, there is:

- The New User's Guide to Editor and Runoff

This guide explains in full detail Prime's editor (ED) and its text formatting utility (RUNOFF). (Aimed at users who may not be programmers, this guide also provides a less technical introduction to Prime software for secretaries, typists and data entry personnel.)

Data Subsystems

POWER is an easy-to-use data management system with English-like commands that allow the user to create, access, update, and report on MIDAS, ASCII, or binary files. POWER files are compatible with (and can be accessed from) BASIC/VM, COBOL, and FORTRAN programs. The guide to using POWER is:

- The PRIME/POWER Guide*

MIDAS - the Multiple Index Data Access System - creates and maintains keyed-index data files to hold large amounts of information in a quickly accessible format. MIDAS files are handled through a variety of high-level language interfaces. Applications programmers working with MIDAS files can consult the:

- MIDAS User's Guide

FORMS allows applications programmers to design screen formats (such as representations of business forms), to store the formats in a directory and to write applications programs that use these screen formats to facilitate data entry. The guide that explains how to do it is:

- The FORMS Guide*

SPSS - a statistical package for the social sciences - is useful to applications programmers who need statistical tools for data handling. The use of SPSS on Prime computers is explained in:

- The SPSS Guide*

Data Base Management

Four guides document Prime's data base management system. Programmers writing data base applications programs in FORTRAN or COBOL should consult:

- The DBMS FORTRAN Reference Guide*
- The DBMS COBOL Reference Guide*

Data base administrators concerned with setting up and maintaining a data base, use:

- The DBMS Administrator's Guide*
- The DBMS Schema Reference Guide*

Communications

If you are installing a network (or if your installation is on a network and you're curious about the details); or if you are writing programs concerned with network functions, the guide you want is:

- The PRIMENET Guide*

If your installation has (or is getting) DPTX (Distributed Processing Terminal Executive), and you're involved with it, you'll want:

- The Distributed Processing Terminal Executive Guide*

If your work involves any of the remote batch terminal emulators - HASP, RJE2780, RJE3780, 200UT, 1004, GRTS, or ICL 7020 - you can find out how to handle them in:

- The Remote Job Entry Guide*

Office Automation

Prime's Office Automation System is currently supported by four documents:

- OAS Word Processing Guide*

  Provides complete instructions for the Word Processing module of Prime's Office Automation System.

- OAS Management Communications and Support Guide*

  Provides instructions for the Management Communications and Support module of Prime's Office Automation System. This module comprises electronic mail, correspondence management and management support functions.

- OAS Advanced Text Management*

  Provides complete instructions for the Advanced Text Management module of Prime's Office Automation System. Advanced Text Management enhances Word Processing by providing automated proofreading and hyphenation, and word-for-word translation in up to four languages is included.

- OAS System Administrator's Guide*

  Provides instructions on management of Prime's Office Automation System. Such items as creation of user ID's, generation and purging of schedule grids and printing hard copies of system reports are included.

PROGRAMMER'S COMPANIONS

Prime also provides a series of handy pocket-sized reference summaries on many of its products. The following titles are currently available:

- FORTRAN: The Programmer's Companion
- BASIC/VM: The Programmer's Companion
- Assembly Language: The Programmer's Companion
- PRIMOS Commands: The Programmer's Companion
- System Administrator: The Programmer's Companion
- Word Processing: The Office Automation Companion

Part II
Writing and Running Programs

SECTION 2

BEFORE YOU GET STARTED

INTRODUCTION

Before you begin using your Prime computer, you'll need to know:

- A few facts about Prime's operating system, PRIMOS
- How to define and organize your files and directories using PRIMOS's file management system
- What the system prompts are
- What conventions Prime guides use when documenting commands
- How to use the terminal
- What meaning the special terminal keys have for the PRIMOS operating system or for some of its subsystems
- What meaning some special characters have for PRIMOS or some of its subsystems
- How to define your own special characters or change the characteristics of your terminal

This section explains all of them, in the above order.

INTRODUCING PRIMOS

All Prime computers, from the 350 up, use a common operating system known as PRIMOS. Under PRIMOS, a Prime computer can support up to 63 simultaneous users. Each user is totally independent. Each one may use any utility (such as an editor or compiler), and may write, compile, load, and execute any program, in any language, without regard to what other users are doing on the system.

Compatibility

Because a common operating system is used throughout the Prime processor line, programs created on one Prime computer can be used on most other Prime computers, without modification. There is complete upward compatibility among all models, and complete downward compatibility among the 750, 650, 550, and 450. Considerable downward compatibility exists among other models as well, as long as system constraints on program size and mode of code generated are observed.

Some Hardware Features

Prime's hardware supports this multi-user, interactive environment with

- Virtual memory, which allows users to run programs larger than the physical memory of the machine. A program may be as large as 32 megabytes on the Prime 450 and up (768 kilobytes on the Prime 350).
- Segmentation of programs, allowing the separation of code and data. This facilitates the creation of pure code for shared or recursive procedures.
- A ring protection system which provides hardware protection for the operating system and user subsystems.

Except for segmentation of large programs, users have little immediate concern with these features. They are largely invisible, designed to let users concentrate on their own goals without worrying about the hardware.

USING THE FILE SYSTEM

File and Directory Structures

A PRIMOS file is an organized collection of information identified by a filename. The file contents may represent a source program, an object program, a run-time memory image, a set of data, a program listing, text of an on-line document, or anything the user can define and express in the available symbols.

Files are normally stored on the disks attached to the computer system. No detailed knowledge of the physical location of a file is required because the user, through PRIMOS commands, refers to files by name. On some systems, files may also be stored on magnetic tape for backup or for archiving.

PRIMOS maintains a separate User File Directory (UFD) for each user to avoid conflicts that might arise in assignment of filenames. A Master File Directory (MFD) is maintained by PRIMOS for each logical disk connected to the system. (A logical disk, sometimes called a volume, may occupy either a complete disk pack or a partition of a multi-head disk pack. In either case, it serves as PRIMOS's basic unit of storage.) The MFD contains information about the location of each UFD on the disk. In turn, each UFD contains information about the location and content of each file or sub-UFD in that directory.

For a description of the PRIMOS file system and a description of the ordering of information within files, refer to The PRIMOS Subroutines Reference Guide.

Pathnames

The PRIMOS file directory system is arranged as a tree. At the root are the disk volumes (also called partitions, or logical disks). Each disk volume has an MFD containing the names of several UFDs. Each UFD may contain not only files, but subdirectories (sub-UFDs), and they may contain subdirectories as well. Directories may have subdirectories to any reasonable level.

A pathname (also called a treename) is a name used to specify uniquely any particular file or directory within PRIMOS. It consists of the names of the disk volume, the UFD, a chain of subdirectories, and the target file or directory. For example,

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

specifies a file on the disk volume FOREST, under the UFD BEECH and the sub-UFD BRANCH5. The file's name is SQUIRREL. Figure 2-1 illustrates how pathnames show paths through a tree of directories and files.

Disk volume names, and the associated logical disk numbers, may be found with the STATUS DISKS command, described later. A pathname can be made with the logical disk number, instead of the disk volume name. For example, if FOREST is mounted as logical disk 3,

```
<3>BEECH>BRANCH5>SQUIRREL
```

specifies the same file as the previous example.

Usually each UFD name is unique throughout all the logical disks. In our example that would mean that there would be only one UFD named BEECH in all the logical disks, 0 through 62. When that is the case, the volume or logical disk name may be omitted, and PRIMOS will search all the logical disks, starting from 0, until the UFD is found. For example, if there is no UFD named BEECH on disks 0, 1, or 2, then

```
BEECH>BRANCH5>SQUIRREL
```

will specify the same file as the previous two examples. This last form of pathname, in which the disk specifier is omitted, is called an ordinary pathname because it is very frequently used.

Pathnames vs Filenames

Most commands accept a pathname to specify a file or a directory. So the terms "filename" and "pathname" may be used almost interchangeably. A few commands, however, require a filename, not a pathname. It is easy to tell a filename from a pathname. A pathname always contains a ">", while a filename or directory name never does.

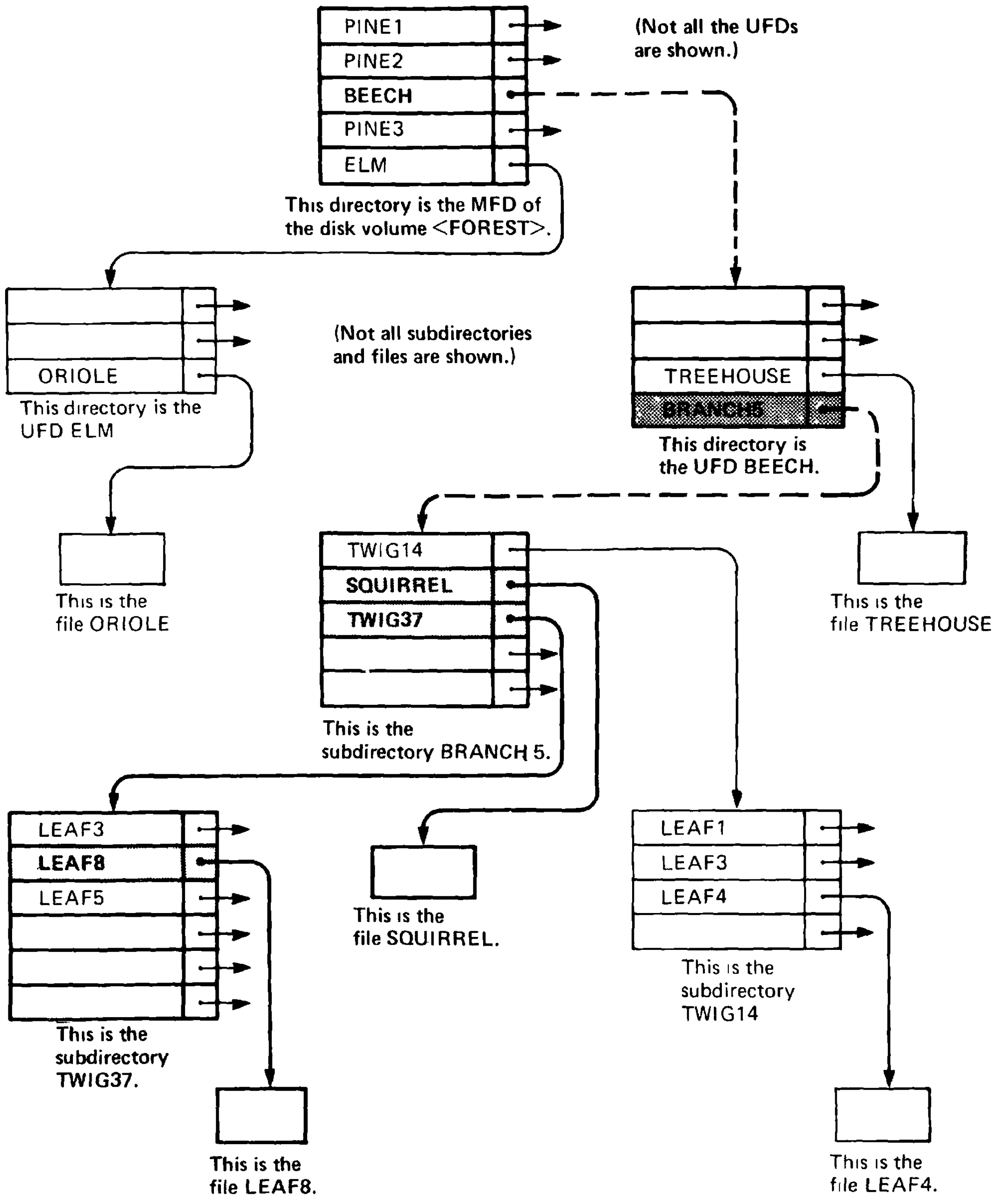

PINE1
PINE2
BEECH
PINE3
ELM
(Not all the UFDs are shown.)
This directory is the MFD of the disk volume <FOREST>.
ORIOLE
This directory is the UFD ELM
(Not all subdirectories and files are shown.)
TREEHOUSE
BRANCH5
This directory is the UFD BEECH.
This is the file ORIOLE
TWIG14
SQUIRREL
TWIG37
This is the subdirectory BRANCH 5.
This is the file TREEHOUSE
LEAF3
LEAF8
LEAF5
This is the subdirectory TWIG37.
This is the file SQUIRREL.
LEAF1
LEAF3
LEAF4
This is the subdirectory TWIG14
This is the file LEAF8.
This is the file LEAF4.

Figure 2-1. Examples of Files and Directories in PRIMOS Tree-structured File System.

Changing Directories

When the user logs in to a UFD, that UFD becomes the home directory. The ATTACH command changes the home directory to any other directory to which the user has access rights. (Thus, the home directory is the directory to which the user is currently attached.) A home directory may be an MFD, UFD, or sub-UFD.

Relative Pathnames

It is often more convenient to specify a file or directory pathname relative to the home directory, rather than via a UFD. For example, when the home directory is:

```
    BEECH>BRANCH5
```

the commands

```
    OK, SLIST BEECH>BRANCH5>TWIG9>LEAF3
```

and

```
    OK, SLIST *>TWIG9>LEAF3
```

have the same meaning. The symbol "*" as the first directory in a pathname means "home directory."

Current Disk

Occasionally it will be necessary to specify a UFD on the disk volume you are currently using; that is, where your home directory is. For example, when developing a new disk volume with UFD names identical to those on another disk, it is necessary to specify which disk is to be used, each time a pathname is given. The current disk is specified by:

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

for example. Do not confuse "<*>", meaning current disk, with the "*" alone, which means home directory.

Passwords

If any directory has a password, the password becomes part of the directory name or pathname. The password is entered after the name of the directory to which it belongs, separated by one blank space. Apostrophes enclose the entire pathname.

For example, if the directory BEECH had a password, SECRET, a pathname using it might be

```
    'BEECH SECRET>BRANCH5'
```

SYSTEM PROMPTS

The OK Prompt

The OK prompt indicates that the most recent command to PRIMOS has been successfully executed, and that PRIMOS is ready to accept another command from the user. The punctuation mark following the "OK" indicates to the user whether he is interfacing with a single-user level of PRIMOS. The prompt "OK:" indicates single-user PRIMOS (a version of PRIMOS II); the prompt "OK," indicates multi-user PRIMOS.

PRIMOS supports type-ahead. The user need not wait for the "OK," after one command before beginning to type the next command. However, since each character echoes as the user types it, output from the previous command may appear on the terminal jumbled with the command being typed ahead. Type-ahead is limited to the size of the terminal input buffer. Default is 192 characters.

PRIMOS II does not support type-ahead. The user must wait for "OK:" before entering the next command.

The ER! Prompt

The ER! prompt indicates that PRIMOS was unable to execute the most recent command, for one reason or another, and that PRIMOS is ready to accept another command from the user. The ER! prompt usually is preceded by one or more error messages indicating what PRIMOS thought the trouble was.

Common errors include:

- Typographical errors
- Omitting a password
- Being in the wrong directory
- Forgetting a parameter or argument

Changing the Prompt Message

Users can change the prompt message displayed at their terminals by using the RDY command. See Section 15, Customizing Your Environment, for details.

CONVENTIONS

All of Prime's user guides and reference guides use a single set of conventions for documenting commands. In all of these guides, the format of a command will be displayed in the following manner:

$$\text{COMMANDNAME} \quad \text{argument} \quad \left\{ \begin{array}{l} \text{-option} \\ \text{-option} \end{array} \right\} \quad \text{[-option]} \quad \ldots$$

The symbols and conventions have the following meanings:

- WORDS-IN-UPPER-CASE

Capital letters identify command words or keywords. They are to be entered literally. (Either upper- or lowercase may be used.) If a portion of an uppercase word is underlined, the underlined letters indicate a system-defined abbreviation.

- Words-in-lower-case

Lowercase letters identify arguments. The user substitutes an appropriate numerical or text value.

- Braces { }

Braces indicate a choice of arguments and/or keywords. At least one choice must be selected.

- Brackets [ ]

Brackets indicate that the keyword or argument enclosed is optional.

- Hyphen -

A hyphen identifies a command line option, as in: SPOOL -LIST. Hyphens must be entered literally.

- Parentheses ( )

When parentheses appear in a command format, they must be included literally.

- Ellipsis ...

The preceding argument or option may be repeated.

- Angle brackets < >

Used literally to separate the elements of a pathname. For example: <FOREST>BEECH>BRANCH537>TWIG43>LEAF4.

- option

The word option indicates that one or more keywords or arguments can be given, and that a list of options for the command follows.

- Spaces

Command words, arguments and options are separated in command lines by one or more spaces. In order to contain a literal space, an argument must be enclosed in single quotes. For example, a pathname may contain a directory having a password:

```
    '<FOREST>BEECH SECRET>BRANCH6'.
```

The quotes ensure that the pathname is not interpreted as two items separated by a space.

Conventions in Examples

In examples of terminal sessions, the user's input will be underlined. The system's prompts or responses will not be underlined. For example:

```
    OK, ATTACH BEECH
    OK,
```

TERMINAL KEYBOARD

Most of the user's interaction with PRIMOS takes place at a computer terminal. Here we review the standard functioning of terminals and present certain aspects unique to Prime.

Basic Layout

The exact layout of the terminal keyboard varies with the type of terminal. Figure 2-2 shows a typical keyboard.

Besides the usual letter, number, and punctuation symbols, the terminal keyboard also has a variety of special symbols and keys. The number and letter keys are arranged in the same positions as on a standard typewriter. The punctuation marks, however, may be located on different keys.

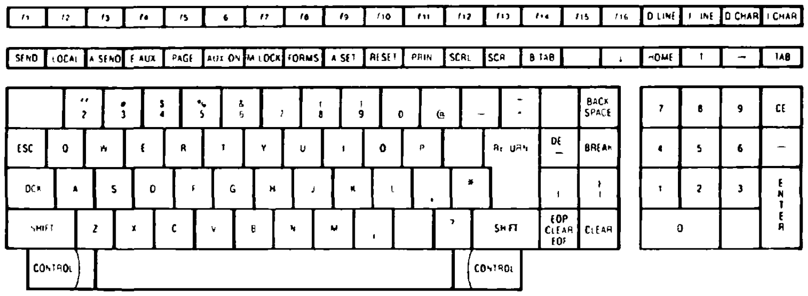


Figure 2-2. Typical Terminal Keyboard

Special keys fall into the following categories:

- Terminal controls and switches
- Special keys
- Special characters

Terminal Controls and Switches

Terminal controls and switches affect the ways a specific terminal performs. Depending on what terminal model is available, these may be on the front, side or bottom of the terminal, or beside the standard keyboard.

The controls and switches of importance are:

ON/OFF: This is the power switch. Some terminals have an indicator light which glows when power is on. On some terminal models, this switch may be located at the rear or on the bottom.

LINE/LOCAL: This switch controls whether or not the terminal is sending input to the computer. In LINE Mode, the terminal and the computer are connected; in LOCAL Mode, the terminal acts like a specialized typewriter. This switch is often labeled: ON-LINE/OFF-LINE, REMOTE/LOCAL or LINE (with an indicator light which is ON in LINE Mode, OFF in LOCAL Mode).

UPPER-CASE/LOWER-CASE: Unlike the SHIFT key, the UPPER-CASE/LOWER-CASE key affects the meanings of the letter keys only. UPPER-CASE causes all letters to print in uppercase, no matter what the setting of the shift key; LOWER-CASE allows selection between upper- and lower-case in the standard manner (by using the shift key).

On some terminals, this switch is located on the bottom, instead of on the keyboard. This key is often labeled: CASE, UPPER/LOWER or U/C (for Upper-Case -- when on). Terminals which produce upper-case letters do not have this switch.

SPECIAL TERMINAL KEYS

- CONTROL

The key labeled CONTROL (or CTRL) changes the meaning of alphabetic keys. Holding down CONTROL while pressing an alphabetic key (or some special keys) generates a control character. Control characters do not print. Some of them have special meanings to the computer. (See CONTROL-P, CONTROL-Q and CONTROL-S, below.)

- RUBOUT

The key labeled RUBOUT has a special use in Prime's text processing utility, RUNOFF. It is not generally meaningful to other standard Prime software. On some terminals it is labeled DELETE or DEL.

- RETURN

The RETURN key ends a line. PRIMOS modifies the line according to any erase (") or kill (?) characters, and either processes the line as a PRIMOS command, or passes it to a utility such as the EDITOR. RETURN is also called CR, CARRIAGE-RETURN, or NEW-LINE.

- $\left.\begin{array}{l}\text{BREAK}\\ \text{ATTN}\\ \text{INTRPT}\end{array}\right\}$ See CONTROL-P

SPECIAL CHARACTERS

- Caret (↑)

Used in EDITOR to enter octal numbers and for literal insertion of special characters. On some terminals and printers, prints as up-arrow (↑).

- Backslash (\)

Default EDITOR tab character.

- Double-quote (")

Default erase character for PRIMOS and all subsystems. Each double-quote erases a character from the current line. Erasure is from right (the most recent character) to left. Two double-quotes erase two characters, three erase three, and so forth. You cannot erase beyond the beginning of a line. The PRIMOS command TERM (described later in this section) allows the user to choose a different erase character.

- Question mark (?)

Default kill character for PRIMOS and all subsystems. Each question mark deletes all previous characters on the line. The PRIMOS command TERM allows the user to choose a different kill character.

- CONTROL-P

QUIT immediately (interrupt/terminate) from execution of current command and return to PRIMOS level. Echoes as QUIT. Used to escape from undesired processes. Will leave used files open in certain circumstances. Equivalent to hitting BREAK key.

- CONTROL-S

Halt output to terminal, for inspection. Program will run until output buffer is full; then it will be suspended. Any commands other than CONTROL-S or CONTROL-Q will be placed in the input buffer (until that buffer is full). They will not execute until the suspended program has terminated. Input will not be echoed at the terminal until either CONTROL-P (QUIT) or CONTROL-Q (Continue) is given. This special function is activated by the command TERM -XOFF.

- CONTROL-Q

Resume output to terminal following a CONTROL-S (if TERM -XOFF is in effect).

- UNDERSCORE (_)

On some devices, prints as a backarrow (←).

- RESERVED CHARACTERS

The following characters are reserved by PRIMOS for special uses. They may not be used in file names:

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

- SEMICOLON (;)

The semicolon is used as a command separator. Using the semicolon, you can place multiple commands on a single line.

SETTING TERMINAL CHARACTERISTICS

Terminal characteristics may be set 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 returns the full list of TERM options available. The format is:

<u>TERM</u> options

The common options are:

Option	Function
-ERASE character	Sets user's choice of erase character in place of the " default.
-KILL character	Sets user's choice of kill character in place of ? default.
-BREAK $\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$	Enables or disables use of CONTROL-P as a BREAK character, to interrupt a running program or command. Default, enabled at LOGIN, is BREAK ON.
-XOFF	Enables X-OFF/X-ON feature, which allows users to suspend terminal output temporarily and to resume it at the point of suspension. Output is halted by typing CONTROL-S and is resumed by typing CONTROL-Q. Also sets terminal to full duplex (default value).
-NOXOFF	Disables X-OFF/X-ON feature (default).
-DISPLAY	Returns list of currently set TERM characters. Also displays current Duplex, Break and X-ON/X-OFF status.

Sending Messages from Your Terminal

You may communicate with users at other terminals by using the MESSAGE command.

The format of the command is:

```
    MESSAGE    [ username   ]      [-NOW]
               [ -usernumber]
```

For complete details on the MESSAGE command see Section 15 of this guide or The PRIMOS Commands Reference Guide.

SECTION 3

ACCESSING PRIMOS

INTRODUCTION

In this section we introduce the essential PRIMOS commands so that you can begin working on the system. We recommend that you keep a Programmer's Companion handy as a summary of the commands explained in this section plus other PRIMOS commands. In this user's guide we have selected only those PRIMOS commands we know will be of use to most programmers. Depending upon your application, there are many other PRIMOS commands that may simplify your task or increase efficiency.

Using PRIMOS

PRIMOS recognizes more than 100 commands, some of which invoke subsystems which themselves respond to subcommands or extensive dialogs. However, most users can do 99 percent of their program development using about a dozen commands. This section introduces the essential commands needed by all users. These commands allow you to:

- Gain admittance to the computer system (LOGIN)
- Change the home directory (ATTACH)
- Create new directories for work organization (CREATE)
- Secure directories against intrusion (PASSWD)
- Remove empty directories or unwanted files (DELETE)
- Examine the location of the home directory and its contents (LISTF)
- Look at the availability and current usage of system resources - space, users, etc. (AVAIL, STATUS, USERS)
- Rename files or directories (CNAME)
- Determine file size (SIZE)
- Examine files (SLIST)
- Allow controlled access to files (PROTEC)
- Complete a work session (LOGOUT)

Table 3-1 summarizes these commands.

Table 3-1. Essential PRIMOS Commands

COMMAND	ACTS ON		PROVIDES		
	FILES	DIRECTORIES	INFORMATION	ACCESS	ACTION
ATTACH		x		x	
CNAME	x	x			x
CREATE		x			x
DELETE	x	x			x
LISTF		x	x		
LOGIN		x		x	
LOGOUT		x		x	
PASSWD		x		x	
PROTEC	x			x	
SIZE	x		x		
SLIST	x		x		

ACCESSING THE SYSTEM

In order to access or work in the system, the user must first follow a procedure known as 'login'. 'Logging in' identifies the user to the system and establishes the initial contact between system and user (via a terminal). Once logged in, the user has access to directories, files and other system resources. The format of the LOGIN command is:

```
    LOGIN ufd-name [password] [-ON nodename]
```

ufd-name	The name of your login directory.
password	Must be included if the directory has a password.
-ON nodename	Used for remote login across PRIMENET network.

For example:

```
    LOGIN DOUROS NIX
    DOUROS (21) LOGGED IN AT 10'33 112878
```

The number in parentheses is the PRIMOS-assigned user number (also called 'job' number). The time is expressed in 24-hour format. The date is expressed as mmddyy (Month Day Year). The word NIX, in this example, is the password on the login directory.

During login, a misspelled UFD will cause the message "Not found. (LOGIN)" to be displayed. A misspelled or incorrect password will return the message "Insufficient access rights. (LOGIN)." If you get either of these messages, check to be sure you're logging into the right directory with the right password; then try logging in again. If you still have trouble, ask your supervisor for help. If the system itself is overloaded, a message such as "maximum number of users exceeded" may be displayed. In this case, log in again later, when some other user may have logged out.

DIRECTORY OPERATIONS

Changing the Home Directory

After logging in, the user's home directory is set to the login UFD by PRIMOS. The user can move (i.e., attach) to another directory in the PRIMOS tree structure with the ATTACH command. The format is:

```
    ATTACH new-directory
```

new-directory is the pathname of the new home directory.

Note

If any directory in the pathname has a password, the pathname must be enclosed in single quotes, as in:

```
A 'BEECH SECRET>BRANCH5'
```

Recovering from Errors While Attaching: If an error message is returned following an ATTACH command (for example, if a UFD is not found), the user remains attached to the previous home directory.

However, if an incorrect password is given, then the user is not attached to any UFD (has no home directory). If a command, such as LISTF, is entered while in this state, the message:

```
NO UFD ATTACHED
```

is returned. To remedy this condition, the user must ATTACH to a UFD as in:

```
A BEECH
```

or to a subdirectory, using a complete or ordinary pathname (but not a relative pathname), as in:

```
A BEECH>BRANCH2
```

Creating New Directories

To organize tasks and work efficiently, it is often advantageous to create new sub-UFDs. These sub-UFDs can be created within UFDs or other sub-UFDs with the CREATE command. They can contain files and/or other subdirectories. The format is:

```
CREATE pathname
```

pathname may be:

- The name of a new subdirectory to be created within the home directory.
- The pathname of a new subdirectory to be created within some other directory.

For example:

```
ATTACH BEECH
CREATE BRANCH6
```

creates the subdirectory BRANCH6 in the directory BEECH.

```
    CREATE ELM>BRANCH1
```

creates the subdirectory BRANCH1 in the UFD ELM.

Two files or sub-UFDs of the same name are not permitted in a directory. If this is inadvertently attempted, PRIMOS will return the message:

```
    Already exists. DIRECTORY-NAME
    ER!
```

Assigning Directory Passwords

Directories may be secured against unauthorized users by assigning passwords with the PASSWD command. There are two levels of passwords: 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. Files can be given different access rights for owners and non-owners with the PROTEC command (see Controlling File Access).

The PASSWD command replaces any existing password(s) on the working 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, if given, follows. If a non-owner password is not specified, the default is null; then, any password (except the owner password) or none allows access to this directory as a non-owner. For example:

```
    OK, A DOUROS NIX
    OK, PASSWD US THEM
```

The old password, NIX, is replaced by the owner password US, and the non-owner password THEM. Passwords may contain almost any characters; but they may not begin with a digit (0-9).

Examining Contents of a Directory

After logging in or attaching to a directory, the user can examine the contents of this directory with the LISTF command which generates a list of the files and subdirectories in the home directory. The format is:

```
    LISTF
```

For example, the working directory is called LAURA. The following list will be generated when LISTF is entered at the terminal:

```
OK, LISTF

UFD=<MISCEL>TEKMAN>LAURA  6 OWNER

$QUERY  BOILER  EX       LETTER  QUERY   OLISTF  BASICPROGS
OUTLINE $OUTLINE         MQL     $MQL    $LETTER MQL.LETTER       FTN10
EXAMPLES         FUTIL.10        $FUTIL.10

OK,
```

The number following the UFD-name is the logical device number, in this case, 6. The words OWNER or NONOWN follow this number, indicating the user status in this directory. (See Assigning Directory Passwords).

If no files are contained in a directory, .NULL. is printed instead of a list of files.

Deleting Directories

When directories or subdirectories are no longer needed, they may be removed from the system to provide more room for current work. If the directories are empty, they may be removed by the DELETE command. The format is:

```
DELETE pathname
```

If an attempt is made to delete directories containing files or subdirectories, PRIMOS prints the message:

```
The directory is not empty. (DIRECTORY-NAME)
```

In this case, the user must do one of two things:

- Use the LISTF command to find what files (or subdirectories) are in the directory. Delete each entry with the command "DELETE filename." Then delete the empty directory.
- Use FUTIL's TREDEL command (explained in Section 10) to delete files and directory simultaneously.

SYSTEM INFORMATION

Table 3-2 summarizes useful information you may need about the system and how to obtain it.

Table 3-2. Useful System Information

Item	Use	PRIMOS commands
Number of users	Indicates system resource usage and expected performance.	STATUS USERS (user list) USERS (number of users)
User login UFD	Identifies user who spooled text file (printed on banner).	STATUS, STATUS UNITS, STATUS ME
User number		STATUS ME, STATUS USERS
User line number		STATUS ME, STATUS USERS
User's physical devices		STATUS ME
Open file units	Avoids conflict when using files.	STATUS, STATUS UNITS
Magnetic tape units	Lists assigned units, with their logical aliases and users	STATUS DEVICE
Disks in operation		STATUS, STATUS DISKS
Assigned peripheral devices	Tells what devices are available.	STATUS USERS
User priorities		STATUS USERS
Other user numbers		STATUS USERS
Your phantom user number	For logging out your phantoms.	STATUS USERS, STATUS ME
Network information	Tells if network is available.	STATUS, STATUS NET
Current nodename		STATUS NET, STATUS UNITS
Records available	Tells how much room is available for file building, sorting, etc.	AVAIL
System time and date	Performs time logging in audit files.	DATE

Table 3-2. (Continued)

Item	Use	PRIMOS commands
Computer time used since login	Measures program execution time.	TIME
Spool queue contents	Tells if job has been printed.	SPOOL -LIST
Names and status of printers	Tells if local printers are functioning.	PROP -STATUS
Environment for a printer	Gives parameters for printer's operations	PROP printer-name -DISPLAY
Batch users	Identifies executing jobs, number of jobs per queue	BATCH -DISPLAY
Your active Batch jobs	Gives job id, status; gives job parameters	JOB -STATUS JOB -DISPLAY
Batch queue status	Lists Batch queues and tells which ones are available for use	BATGEN -STATUS
Batch queue configurations	Shows environment of Batch system	BATGEN -DISPLAY

Note

Information given by any STATUS command is also given by the STATUS ALL command.

FILE OPERATIONS

Creating and Modifying Files

Text files are created and modified using the text editor (ED). They are printed on the line printer using the SPOOL command. Both these processes are discussed in Section 4. Files may be transferred from other systems not connected via PRIMENET using magnetic tape (MAGNET command), paper tape (ED command), or punched cards (CRSER command). These commands are described in Section 11.

Changing File Names

It is often convenient or necessary to change the name of a file or a directory. This is done with the CNAME command. The format is:

```
    CNAME old-name new-name
```

old-name is the pathname of the file to be renamed, and new-name is the new filename. For example:

```
    cn tools>more_test oldtest
```

The file named MORE_TEST in the UFD TOOLS is changed to OLDTEST. Since no disk was specified, all MFDs (starting with logical disk 0) are searched for the UFD TOOLS.

If new-name already exists, PRIMOS will display the message:

```
    Already exists.  OLDTEST
    ER!
```

An incorrect old-name prompts the message:

```
    Not found. MORETEST
    ER!
```

Determining File Size

The size (in decimal records) of a file is obtained with the SIZE command. This command returns the number of records and words in the file specified by the given pathname. The number of records in a file is defined as the total number of data words divided by 440. However, a zero-word length file always contains one record. The format is:

```
    SIZE pathname
```

For example:

```
    OK, SIZE DATA.FILE.1
        2Ø5 RECORDS IN FILE (89762 words)
```

Examining File Contents

Contents of a program or any text file can be examined at the terminal with the SLIST command. The format is:

```
    SLIST pathname
```

The file specified by the given pathname is displayed at the terminal. It is possible to suspend the terminal display as it is printing. See the discussion on TERM, in Section 2.

Deleting Files

When files or programs are no longer needed they may be removed from the system to provide more room for other uses. The DELETE command deletes files from the working directory. The format is:

```
    DELETE pathname
```

SEG runfiles cannot be deleted by this command. They must be deleted by SEG's own delete command (explained in Section 6) or by FUTIL's TREDEL command (explained in Section 1Ø).

Controlling File Access

Assigning passwords to directories allows users working in a directory to be classified as owners or non-owners, depending upon which password they use with the ATTACH command. Controlled access can be established for any file using the PROTEC command. This command sets the protection keys for users with owner and non-owner status in the directory. (See Assigning Directory Passwords above.) The format is:

```
    PROTEC pathname [owner-rights] [non-owner-rights]
```

pathname	The name of the file to be protected.
owner-rights	A key specifying owner's access rights to file (original value = 7).
non-owner-rights	A key specifying the non-owner's access rights (original value = Ø).

The values and meanings of the access keys are:

Key	Rights
0	No access of any kind allowed
1	Read only
2	Write only
3	Read and Write
4	Delete and truncate
5	Delete, truncate and read
6	Delete, truncate and write
7	All access

For example:

```
    PROTEC <OLD>MYUFD>SECRET 7 1
```

In this example, protection rights are set on the file SECRET in the UFD MYUFD so that all rights are given to the owner and only read rights are given to the non-owner.

Note

The default protection keys associated with any newly created file or UFD are: 7 0. The owner is given ALL rights and the non-owner is given none. Default values for the PROTEC command, however, are: 0 0. Thus, the command PROTEC MYFILE denies all rights to owner and non-owner alike.

COMPLETING A WORK SESSION

When finished with a session at the terminal, give the LOGOUT command. The format is:

```
    LOGOUT
```

PRIMOS acknowledges the command with the following message:

```
    UFD-name (user-number) LOGGED OUT AT (time) (date)
    TIME USED = terminal-time CPU-time I/O-time
```

user-number	The number assigned at LOGIN.
terminal-time	The amount of elapsed clock time between LOGIN and LOGOUT in hours and minutes.
CPU-time	Central Processing Unit time consumed in minutes and seconds.
I/O-time	The amount of input/output time used in minutes and seconds.

It is a good practice to log out after every session. This closes all files and releases the PRIMOS process to another user. However, if you forget to log out, there is no serious harm done. The system will automatically log out an unused terminal after a time delay. This delay is set by the System Administrator (the default is 1ØØØ minutes but most System Administrators will lower this value).

SECTION 4

CREATING SOURCE FILES

ENTERING AND MODIFYING PROGRAMS -- THE EDITOR

Programs are normally entered into the computer using Prime's Text EDITOR (ED). This EDITOR is a line-oriented text processor. That is, it enters and modifies text on a line-by-line basis, keeping track of its current location by a line pointer that is always located at the last line processed (whether the processing action is printing, locating, moving pointer, etc.). The EDITOR operates in two modes, INPUT and EDIT.

Using the EDITOR

When creating a new file, the EDITOR is invoked by

```
ED
```

which places the EDITOR in the INPUT mode. When modifying an existing file, the EDITOR is invoked by

```
ED filename
```

which places the EDITOR in the EDIT mode.

A RETURN with no preceding characters on that line switches the EDITOR from one mode to another.

Input Mode

The INPUT mode is used when entering text information into a file (e.g., creating a program). The word INPUT is displayed at the user's terminal to indicate that the EDITOR has entered that mode. The RETURN key terminates the current line and prepares the EDITOR to receive a new line. Tabulation is done with the backslash (\) character. Each backslash represents the first, second, etc., tab setting; the default tabs are at columns 6, 15, and 30. These settings may be overridden and up to 8 tab settings may be specified by the user with the TABSET command (described in Appendix E). A RETURN with no text preceding it puts the EDITOR into EDIT mode.

Edit Mode

The EDIT mode is used when the contents of the file are to be modified. More than 50 commands are available, although users will find that a small subset of these will suffice for most purposes. The commands in this subset are listed and described in detail later in this section. For a complete list of commands, see Appendix E.

In EDIT mode, the EDITOR maintains an internal line pointer at the current line (the last line processed). Commands such as TOP, BOTTOM, FIND, and LOCATE, move this pointer. WHERE prints out the current line number; POINT moves the pointer to a specified line number. The MODE NUMBER command causes the line number to be printed out whenever a line of text is printed. All commands for location and modification begin processing with the current line.

A RETURN without any preceding characters puts the EDITOR into the INPUT mode.

Special Characters

In either mode, a single character can be erased with the erase character (default is "). For each " typed, a character is erased (from right to left). The entire current line may be deleted by typing the kill character (default is ?). A line followed by a ? is null, and a RETURN at that point will switch the EDITOR into the other mode.

In input mode, the semicolon (;) is equivalent to a CR (ends a line of input). In edit mode, semicolons in a character string are treated as a printing character; semicolons within commands separate multiple commands entered on the same line. A special character may be entered literally in either mode by preceding it with an escape character (^). Special characters may be changed using the TERM command (explained in Section 2).

Saving Files

Orderly termination of an EDITOR session is done from EDIT mode. The command:

```
    FILE filename
```

writes the current version of the edited file to the disk under the name filename. The specified file will be created if it did not previously exist or overwritten if it does exist. If an existing file is being modified, the command

```
    FILE
```

writes the edited version to the disk with the old filename. After execution of the filing command, control is returned to PRIMOS.

A file may also be saved without leaving the EDITOR. The command:

```
    SAVE filename
```

saves a file in its current state under filename. The filename may be the original name of the file or a new name. You may then resume editing your current file. If filename is not specified, the current filename is used. The name of the file is then printed on the terminal screen.

Useful Techniques

The following will aid the user in adapting to Prime's EDITOR:

Tab Settings: When entering source code, much time can be saved using the TABSET command. In INPUT mode, each \ character is interpreted as one tab setting; the default values are columns 6, 15, and 3Ø. Tabs may be set to whatever values each programmer finds useful. Setting a tab near column 45 makes entry of in-line comments simple; the use of such comments in programs is strongly advised.

Moving Lines of Code: Any number of lines can be moved from one location to another using the DUNLOAD command. DUNLOAD deletes these lines as it writes them into an auxiliary file. A LOAD command loads the new file at the desired point. Any number of lines can be copied from one location in a program to another using the UNLOAD command. UNLOAD does not delete the lines as it writes them into an auxiliary file. A LOAD command loads the copy from the new file at the desired point.

Overlaying Comments After Code is Written: Comments may be easily added to an existing source program with the OVERLAY command in conjunction with the TABSET command.

Finding a Line by Label or Statement Number: The FIND command may be used to locate a statement number in a FORTRAN program or a label in a COBOL or PL/I program.

Modifying a Line Without Changing Character Positions: The MODIFY command is used when a line must be modified but the absolute column alignment must remain the same.

Column Display: Entering source code and other data is facilitated by the column display feature. A banner of column numbers is displayed on the terminal for an alignment guide. The MODE COLUMN command, given in Edit mode, causes this display to be printed each time Input mode is entered during an EDITOR session.

Other Hints: When entering FORTRAN programs, it is often helpful to use the TABSET command to reset tabs to columns 7 and 45.

When entering PL/I programs, use the SYMBOL command to change the SEMICO character (which normally tells the EDITOR of the end of a line or command) from a semicolon to something else. For example:

```
    SYMBOL SEMICO {
```

In this example, the brace becomes the EDITOR's line-ending symbol, and the semicolon is freed for its PL/I functions.

To enter a single semicolon (or other special character), precede it with an up-arrow.

To enter up-arrows literally, type two up-arrows. The result displays as two up-arrows on the terminal, but prints as one up-arrow on the printer and is interpreted as a single up-arrow by compilers.

EDITOR'S ERROR MESSAGES

In edit mode, if you give EDITOR a command that it cannot understand, you will receive one of the following error messages:

- BAD abbreviator -- This means you did not use the proper format for the command.
- ? -- Your input could not be interpreted as any of the EDITOR commands. This is often a result of thinking that you are in input mode when you are still in edit mode.

BASIC EDITOR COMMANDS

You should be able to do most of your text-editing using the following selection of the EDITOR commands:

The PRINT Command

Location Commands
- TOP
- BOTTOM
- NEXT
- POINT

String-Finding Commands
- LOCATE
- FIND
- NFIND
- FIND(n)
- NFIND(n)

Text-Changing Commands
- APPEND
- CHANGE
- DELETE
- INSERT
- IB
- RETYPE
- OOPS

DUNLOAD
LOAD
UNLOAD

Ending and Saving an EDITOR Session Commands
QUIT
FILE
SAVE

These commands are discussed in detail in the following few pages. All of the EDITOR commands are listed in Appendix E of this guide and in The New User's Guide to EDITOR and RUNOFF.

Note

The string argument in the commands in this section are 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 (^).

Valid command abbreviations are underlined.

Sample File

The following FORTRAN program is used in all the examples in this section.

```
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
           LNUMB (I) = I
           WRITE (1, 200) LNUMB (I)
200   FORMAT (10X, I5)
100   CONTINUE
      CALL EXIT
      END
```

The PRINT Command

The PRINT command prints n lines of your file, including the current line, and makes the last line PRINTed the new current line. The format of the PRINT command is:

```
PRINT [n]
```

n is the number of lines you want printed. If n is -1, 0, or omitted, the default value is 1. If n is negative, EDITOR moves the pointer back n lines from the current line, and then prints one line, which is the new current line. For example:

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

The space between PRINT and n is optional. A PRINT immediately after the following commands yields .NULL.: TOP, BOTTOM, DELETE, DUNLOAD, LOAD.

Location Commands

Location commands move the pointer to a specific line. EDITOR's specific location commands are TOP, BOTTOM, NEXT, and POINT.

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

```
TOP
```

Example:

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

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

```
BOTTOM
```

Example:

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

```
PRINT 5
      CALL EXIT
      END
BOTTOM
```

The NEXT Command: The NEXT command moves the pointer n lines and prints the new current line. Positive values of n move the pointer down towards the bottom of the file; negative values move the pointer up towards the top. The format of the NEXT command is:

```
NEXT [n]
```

If n is Ø or unspecified, the default value of 1 is used. 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 printed. For example:

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

The POINT Command: The POINT command positions the pointer at line n. The line numbers are not actually part of your file; EDITOR generates them for its own reference. The format of the POINT command is:

```
POINT n
```

The POINT command is equivalent to the sequence TOP, NEXT n. The value of n must be greater than Ø. POINT Ø will give you an error message. POINT 1 is equivalent to TOP, NEXT. If n is greater than the number of lines in the file, the pointer will be left at the bottom. For example:

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

String-Finding Commands

The LOCATE and FIND commands reposition the pointer to the first line below the current line containing the specified string. The NFIND command repositions the pointer to the first line below the current line which does not begin with the specified string.

These commands distinguish between uppercase and lowercase letters in a specified string. If you are unable to find old lines in your file, but can find newly inserted ones, and your current display is set for all CAPS, the CASE control on your terminal may be in the wrong position.

The LOCATE Command: The LOCATE command locates the first line below the current line which contains string anywhere in that line and prints the line on your terminal. The format of the LOCATE command is:

```
    LOCATE string
```

If no line containing string is found below the current line, BOTTOM will be printed and the pointer left at the end of the file. The string cannot contain commas.

Example:

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

The FIND Command: The FIND command is a specialized version of the LOCATE command. It searches for a string and prints the string when found. The string, however, must begin in column one in order for FIND to locate the string. The format of the FIND command is:

```
    FIND string
```

If no line beginning with string is found, the pointer stops at the end of the file, and the word BOTTOM is printed. The string cannot contain commas.

Example:

```
    FIND C
    C This program generates the numbers 1 to 1Ø
    FIND 1ØØ
    1ØØ   CONTINUE
```

The NFIND Command: The NFIND command moves the pointer to the first line below the current line which does not begin with string. The format of the NFIND command is:

```
    NFIND string
```

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: You can also find a string starting in a column other than column 1 by specifying the number of the column within parentheses directly after the command word.

```
FIND(n) string
```

The parentheses () around the column number are required. There cannot be 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)
```

Like FIND, you can NFIND beginning on a column other than column 1 using the format:

```
NFIND(n) string
```

Example:

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

Text-Changing Commands

The APPEND, CHANGE, DELETE, INSERT, IB, RETYPE, OOPS, DUNLOAD, LOAD, and UNLOAD commands alter the text on one or several 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
```

Remember: One blank separates the command word APPEND (or abbreviation) from the string you wish to append. All further blanks are treated as part of the string.

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 first character after the command word CHANGE (or abbreviation) is used as the delimiter. The format of the CHANGE command is:

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

Example:

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

Use a delimiter which does not occur in the text you are changing. Slash is a common delimiter, but if your text to be changed contains slashes, use a different character, as in this example:

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

If the letter G (for General) is specified, CHANGE will change every occurrence of string-1 on a line. If you do not specify G, only the first incidence of string-1 will be changed.

If the value of n is either 0 or 1, EDITOR only makes changes on the current line. (If n is either 0 or unspecified, the default value of 1 is used.) If a value other than 0 or 1 is specified, EDITOR will inspect and make changes on n lines starting at the current line, and leave the pointer positioned at the nth line. If there are fewer than n lines in the file the message BOTTOM is printed. EDITOR prints out all changed lines, plus the last line examined.

Note

1. Remember to issue the TOP command before making changes on the file as a whole.

2. If you end the command with a RETURN, you can omit the closing delimiter.

3. You can specify the semicolon (;) as a text character within the delimiters -- e.g., if you used "@" every place in your file where you wanted to use ";" then the command sequence TOP, CHANGE/@/;/G9999 would change all the @'s to ;'s. (Make sure n is greater than the number of lines in your file.)

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

    ```
    CHANGE//string/
    ```

    Example:

    ```
    LNUMB (5Ø), LSTORE (5Ø)
    CHANGE//      DIMENSION /
          DIMENSION LNUMB (5Ø), LSTORE (5Ø)
    ```

The DELETE Command: The DELETE command deletes n lines, including the current line, and leaves the pointer at the null line where the last deleted line was. The null line is maintained, in case you wish to insert a new line, until a new command moves the pointer away. The format of the DELETE command is:

```
DELETE [n]
```

If n is not specified, the default value of 1 is used. n may be positive or negative, indicating deletion of the current line plus n-1 lines below or above the current line. Since n always indicates the current line, the commands d, d1 and d-1 are all equivalent.

Example:

```
TOP
PRINT 5
.NULL.
C This program generates the numbers 1 to 1Ø
C and prints the numbers on the terminal screen.
C
      DIMENSION LNUMB (5Ø), LSTORE (5Ø)
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 1Ø
C
      DIMENSION LNUMB (5Ø), LSTORE (5Ø)
```

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

```
INSERT newline
```

Example:

```
      DIMENSION LNUMB (5Ø), LSTORE (5Ø)
        DO 1ØØ I = 1, 1Ø
NEXT -1
      DIMENSION LNUMB (5Ø), LSTORE (5Ø)
INSERT        COMMON LSTART (5Ø)
PRINT 2
        COMMON LSTART (5Ø)
        DO 1ØØ I = 1, 1Ø
```

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

```
IB newline
```

Example:

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

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

```
RETYPE string
```

Remember: The first space after RETYPE separates the command word from the parameter; all further spaces are part of string.

Example:

```
C Thsi porgarm gennratse hte mumbers 1 too 1999
C and prints the numbers on the terminal screen.
C
NEXT -2
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
```

The string is terminated by either a semicolon (;) or a RETURN.

RETYPE followed immediately by a space and a RETURN erases the current line and replaces it with a blank line; RETYPE followed by a RETURN yields: BAD RETYPE.

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
BAD RETYPE
RETYPE 
PRINT

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 undoes the last line changed and reinstates it to its condition before the modification. This command does not work for changes to multiple lines at a time. The format of the OOPS command is:

```
OOPS
```

Example:

```
          DIMENSION LNUMB (5Ø), LSTORE (5Ø)
    CHANGE/DIMENSION/COMMON
          COMMON LNUMB (5Ø), LSTORE (5Ø)
    OOPS
          DIMENSION LNUMB (5Ø), LSTORE (5Ø)
```

The DUNLOAD Command: The DUNLOAD command creates a new file with indicated filename, copies n lines from the EDITOR work file, beginning with the current line, into this new file, and then deletes these n lines from the work file. The format of the DUNLOAD command is:

```
    DUNLOAD filename [n]
```

If filename is not specified, you will get the error message: BAD DUNLOA. However, be careful not to specify a filename currently in use unless you want the old file wiped out.

If n is not specified, the default value of 1 is used and one line is DUNLOADed. DUNLOAD leaves the pointer positioned at a null line where the deleted lines used to be; this null line disappears as soon as the pointer is moved.

The DUNLOAD command is useful for moving lines of text to different places; DUNLOAD can also be used instead of DELETE if you want to make sure you don't accidentally delete large blocks of text.

Example:

```
    TOP
    PRINT 12
    .NULL.
          DIMENSION LNUMB (5Ø), LSTORE (5Ø)
            DO 1ØØ I = 1, 1Ø
               LNUMB (I) = I
               WRITE (1, 2ØØ) LNUMB (I)
    C This program generates the numbers 1 to 1Ø
    C and prints the numbers on the terminal screen.
    C
    2ØØ   FORMAT (1ØX, I5)
    1ØØ   CONTINUE
          CALL EXIT
          END
    NEXT -6
    C This program generates the numbers 1 to 1Ø
    DUNLOAD COMMENTS 3
    TOP
    PRINT 12
    .NULL.
          DIMENSION LNUMB (5Ø), LSTORE (5Ø)
            DO 1ØØ I = 1, 1Ø
               LNUMB (I) = I
               WRITE (1, 2ØØ) LNUMB (I)
```

```
200   FORMAT (10X, I5)
100   CONTINUE
      CALL EXIT
      END
BOTTOM
```

The LOAD Command: The LOAD command copies the contents of a file (specified by its filename) into the EDITOR work file just below the current line. The pointer will then be just below the end of the LOADed text, positioned at a null line. The format of the LOAD command is:

```
LOAD filename
```

LOAD does not affect the contents of the original file filename in any way; it simply copies the contents of filename into the work file.

Example:

```
TOP
PRINT 3
.NULL.
      DIMENSION LNUMB (50), LSTORE (50)
        DO 100 I = 1, 10
TOP
LOAD COMMENTS
EDIT
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
```

Note

LOADed text will not go in your permanent files in your UFD unless you FILE at the end of the EDITing session. (The FILE command is discussed in detail later in this section.)

The UNLOAD Command: The UNLOAD command copies n lines beginning at the current line from the file being EDITed into a new file named filename. The format of the UNLOAD command is:

```
UNLOAD filename [n]
```

If n is 0 or omitted, it is assumed to be 1. A negative value for n UNLOADs the preceding n-1 lines and the current line, in the correct order.

The last line UNLOADed is the new current line. Be careful not to specify a filename currently in use unless you want the old file wiped out.

UNLOAD does not delete the lines of the work file as it writes these lines into the file filename.

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
```

Ending and Saving an EDITOR Session

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

The QUIT Command: The QUIT command tells the EDITOR you do not want to save the EDITOR work file, instead want to preserve the original and want to return to PRIMOS-level. The format of the QUIT command is:

```
QUIT
```

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

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

This message asks whether EDITOR may throw away the work file.

A YES (or Y, YE, O, OK, or NULL line RETURN) response returns you to PRIMOS without saving the current session's editing. Any other response provokes a PLEASE FILE message (see the explanation of the

FILE command). If you did not create or modify a file, saying QUIT automatically returns you to PRIMOS.

The FILE Command: The FILE command turns the EDITOR work file (which is so far only a temporary file) into a permanent file in your UFD and returns you to PRIMOS.

WARNING

> Since the work file does not exist outside of EDITOR, you must FILE if you want to save your work. If you do not FILE or SAVE your work, it will be destroyed.

The format for the FILE command is:

```
FILE [filename]
```

If you have been creating a new file, you must specify filename. (The error message FILENAME MUST BE SPECIFIED occurs if you don't.)

You cannot have two files with the same name in the same UFD! If you give a filename which already exists in your UFD, EDITOR will delete the old file of that name from your UFD (without any warning), and put the EDITOR work file in its place.

The same warning holds true for old files. If you have been working on an old file, and you specify the old filename, or say FILE without any filename, your old copy will be deleted, and your new version kept. Giving a new filename keeps both the old and new versions.

The rules for making filenames are:

1. Filenames can be up to 32 characters long.

2. Filenames can contain only the following characters: A through Z, Ø through 9, & - $ * . _ / #

3. The first character may be any legal character except a digit.

4. Characters NOT permitted in filename include: imbedded blanks and special characters such as ? ! @ ; , "

5. Uppercase and lowercase letters are treated as uppercase by PRIMOS. (Letters entered in lowercase are converted to uppercase.)

Valid Filenames	Invalid Filenames
NEWFILE	A?
Todays-Prices	Two@John
Highs&Lows	"Eureka"
$monthly.REPORT	Why a Duck
R34587	1file
A-tale-of-two-cities	

The FILE command can also be used to make copies of any file, simply by typing ED plus filename and FILEing the copied work file immediately with a new filename, as in:

```
OK, ED FTN.TEST
EDIT
FILE TEST

OK,
```

The SAVE Command: The SAVE command writes the contents of the current EDITOR session into filename but does not leave the EDITOR or terminate the current session. The format of the SAVE command is:

```
SAVE filename
```

filename is the name of the file you want the current session copied to. If filename is not specified, the current EDITOR session is written into the filename being edited.

SAMPLE EDITING SESSIONS

Here are three examples showing the writing and editing of source files.

A PL/I Example

```
OK, ED
INPUT
                                  An empty line puts us in edit mode
EDIT
TABSET 3 6 9 12 18 21             Set tabs for PL/I code
SYMBOL SEMICO {                   Change EDITOR's delimiter symbol
                                  Empty line puts us in input mode
INPUT
\DO I=1 TO 10                     Type in source code using tabs
\\DO J=1 TO 10;                     to show levels of indentation
\\\X(I,J=A(I)+B(I);
\\\Y(I,J)=SQRT(X(I,J));
\\END;  /*J-LOOP*/
\END;  /*I-LOOP*/

EDIT
TOP                               Go to top of file
NEXT                              First non-null line
  DO I=1 TO 10
APPEND ;                          Add a forgotten semicolon
  DO I=1 TO 10;
NEXT 2                            Down two lines
        X(I,J=A(I)+B(I);
CHANGE/J/J)                       Balance parentheses
        X(I,J)=A(I)+B(I);
TOP
PRIT ?PRINT 99                    Check code before filing
.NULL.
  DO I=1 TO 10;
     DO J=1 TO 10;
        X(I,J)=A(I)+B(I);
        Y(I,J)=SQRT(X(I,J));
     END;  /*J-LOOP*/
  END;  /*I-LOOP*/
BOTTOM
FILE ED.EX                        Name a new file when you file it

OK,
```

A FORTRAN Example

```
OK, ED
INPUT

EDIT
TABSET 7 45                              Useful settings for FORTRAN

INPUT
\A-"=3Ø\/* COMMENT      Quote mark erases one character
\B=4Ø
C-A?\C=A+B       Question mark erases entire line.
\PRINT 1Ø,C
CALL EXTI""IT
\END

EDIT
FILE FTN.TEST

OK, ED FTN.TEST
EDIT
PRINT 2Ø
.NULL.
      A=3Ø                                  /* COMMENT
      B=4Ø
      C=A+B
      PRINT 1Ø,C
CALL EXIT
      END
BOTTOM
NEXT -3                                     Move up three lines
      PRINT 1Ø,C
TABSET 7 45                                 Set FORTRAN tabs again
INSERT 1Ø\FORMAT('THE ANSWER IS',I4)        Insert forgotten line
TOP, PRINT 2Ø                               Check file once more
.NULL.
      A=3Ø                                  /* COMMENT
      B=4Ø
      C=A+B
      PRINT 1Ø,C
1Ø    FORMAT('THE ANSWER IS',I4)
CALL EXIT
      END
BOTTOM
FILE                         No need to use a filename this time
FTN.TEST

OK,
```

A COBOL Example

```
OK, ED
INPUT

EDIT
MODE COLUMN

INPUT
         1         2         3         4         5         6         7
1234567890123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789
       ID DIVISION.
       PROGRAM-ID.  TEST.               Source coding is keyed in,
       INSTALLATION.  PRIME.            aligned by column.
\ *
                                        The first tab default is position
                                        6.  A space after the backslash
                                        character positions the asterisk
                                        in the continuation column 7.


EDIT
FILE COBOL.TEST

OK, ED COBOL.TEST
EDIT
PRINT 23
.NULL.
       ID DIVISION.
       PROGRAM-ID.  TEST.
       INSTALLATION.  PRIME.
      *


BOTTOM
FILE
COBOL.TEST

OK,
```

LISTING PROGRAMS

Terminal Listing

Source programs may be listed at the terminal by using the SLIST command, described in Section 3.

Line Printer Listing

Use the SPOOL command (explained below) to obtain a copy of a source file on the system line printer.

Renaming

Programs may be renamed with the PRIMOS command CNAME (Section 3). You must have owner status in the UFD in order to use this command.

Deleting

Programs may be deleted with the PRIMOS command DELETE (Section 3). You must have delete access in order to use this command.

PRINTING FILES WITH THE SPOOL COMMAND

Printed copies of files from a line printer are obtained with the SPOOL command. It has several options, some of which will not apply to all systems, as systems may be configured differently. The format is:

```
    SPOOL pathname [options]
```

PRIMOS makes a copy of pathname in the Spool Queue for the line printer, and displays the message:

```
    Your spool file,  PRTnnn, is     x record[s] long.
```

nnn is a 3-digit number which identifies the file in the Spool Queue. x is number of records in the file. PRIMOS spools out short files as soon as possible; long files receive a lower priority. For example:

```
    OK, spool example
    [SPOOL rev 17.2]
    Your spool file,  PRT015, is     2 records long.

    OK, spool tekman>alice>update
    [SPOOL rev 17.2]
    Your spool file,  PRT016, is     1 record  long.

    OK,
```

In this example, one file was spooled by filename and the other by pathname. However, SPOOL will refer to both by their filenames, that is, EXAMPLE and UPDATE.

Checking the Queue

To check the status of the Spool Queue, give the command:

```
    SPOOL -LIST
```

PRIMOS returns a list of all the files on the Queue which have not yet been printed. Additional information, such as the size, destination, the PRT number, any options, the form-type and the login-name of the user who spooled the file, are also specified. For example:

```
    OK, spool -list
    [SPOOL rev 17.4]

     user   prt  time  name              size  opts/#  form   defer  at: CAROUSEL
    ------ ---  -----  ---------------- -----  ------  ------ -----  -------------
    BLAISE 006  11:08  VISTA-REPORT        2
    GRACE  007   9:31  $S4.4303           44           WHITE         1
    TEKMAN 008   9:32  $S7.4130            9           WIDE          1
    ALICE  009   9:32  INDEX               1           NOW
```

Cancelling a Spool Request

To cancel one or more spool requests, the command format is:

```
    SPOOL -CANCEL [PRT]n-1 [,n-2...]
```

where n-1, n-2, etc., are the numbers of your spool files to be cancelled. For example:

```
    OK, spool -cancel 47 048 prt049
    [SPOOL rev 17.0]
    PRT047 has been cancelled.
    PRT048 has been cancelled.
    PRT049 has been cancelled.
```

Printing Multiple Copies

You can request several copies of one file by using the -COPIES option:

```
    SPOOL pathname -COPIES n
```

n is the number of copies desired.

Deferring Printing

The -DEFER option tells the Spooler not to begin printing the indicated file until the system time matches the time specified with DEFER. This permits you to enter SPOOL requests at your convenience, rather than waiting for the appropriate hour.

Specify the DEFER option by:

```
    SPOOL pathname -DEFER time
```

The format for time is HH [:] MM [AM/PM]. If AM or PM is given, HH:MM (the colon is optional) must be in 12-hour format (e.g., 1ØØØ PM). Otherwise, time will be interpreted as 24-hour format (in which 22ØØ is 1Ø:ØØ PM and 1ØØØ is 1Ø:ØØ AM).

Printing on Special Forms

Line printers traditionally use one of two types of paper -- "wide" listing paper, on which most program listings appear, and 8-1/2 x 11-inch white paper, which is standard for memos and documentation. Computer rooms often stock a variety of special paper forms for special purposes, such as 5-copy sets, pre-printed forms (checks, orders, invoices), or odd sizes or colors of paper.

Request a specific form by:

```
    SPOOL pathname -FORM form-name
```

form-name is any six-character (or less) combination of letters. A list of available form names can be obtained with the PROP command, explained in the PRIMOS Commands Reference Guide.

Changing the Header

The -AS option tells the spooler to print your file under a different name. The form is:

```
    SPOOL pathname -AS alias
```

The alias will appear on the header and in the SPOOL -LIST display.

Printing at Specific Locations

Networks with several printers often arrange to have the printers read each other's queues. It is therefore possible for a spool request to be printed at another location, perhaps many miles distant. To insure that a spool request is printed where you want it, use the -AT option:

```
    SPOOL pathname -AT destination
```

destination is a word of 16 letters or less. A list of available destination-names can be obtained with the PROP command, explained in the PRIMOS Commands Reference Guide. (If a destination appears in the heading of the SPOOL -LIST display, for example, at: CAROUSEL, then that destination is the default destination for spool requests. If no destination folows "at:", then no default has been established, and spool requests without destinations may be intercepted by any available printer.

Eliminating Headers

To have files printed without header or trailer pages, use the -NOHEAD option:

```
SPOOL pathname -NOHEAD
```

This option is particularly useful with preprinted forms, but if you're using this option in a multi-user environment, you will have to identify your own jobs.

Multiple Options

Any or all of the above options may be used jointly in a single SPOOL command line. If -LIST or -CANCEL is included, it must be the last option on the command line. For example:

```
OK, spool o_17 -as ex.1 -at bldg.1 -defer 22:00
[SPOOL rev 17.0]
Your spool file, PRT048, is     1 record  long.
```

This particular command requests that the file named "O_17" be printed at the "bldg.1" printer, under the alias of "EX.1", at 10 pm (22:00).

PRINTING SEVERAL FILES IN ONE WITH THE CONCAT COMMAND

The CONCAT command concatenates files into a single file, which can then be printed via the SPOOL command. The format for CONCAT is:

```
CONCAT new-filename [-options]
```

Options govern the format of the print-out and the disposition of the files. For details, see 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 prints a colon prompt. Type the filenames, one per line. A null line (carriage return) signals the end of list. CONCAT then goes into command mode, and prints a right-angle prompt. You can then type a QUIT to end the session. (You can also type "INPUT" to return to input mode; or you can give various formatting commands, explained in the

PRIMOS Commands Reference Guide.)

A sample session might be:

```
    OK, concat triplet
    [CONCAT Rev 17.0]

    Enter filenames, one per line:
    : first
    : second
    : third
    : (CR)

    > q

    OK,
```

If the file TRIPLET already exists, CONCAT asks:

```
    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 one. Answering APPEND preserves the existing contents of TRIPLET and adds the new ones at its end.

SECTION 5

COMPILING PROGRAMS

After the source code has been entered into the system, it must be compiled. Compilation creates a new file of linkable code--the object (or binary) file. Each high-level language has its own compiler which creates object code from source code. At the object code level, these languages are equivalent. Thus, modules of object code originating from different source languages may be linked together to form a run-time program. (Further comments on this will be found at the end of this section.) Details of Prime's compilers are treated in the individual language user guides. This section will consider features common to all compilers.

Prime's convention for source file names uses compiler-name suffixes. The suffixes are separated from the "base name" of the file by a dot. Thus, a FORTRAN IV program with a base name DRAGON would be named DRAGON.FTN, while a FORTRAN 77 program with a base name WYVERN would be named WYVERN.F77. This naming convention helps you keep track of the types of files in your directories. It also helps you access groups of files using CPL's WILD function. (See The CPL User's Guide for details.)

Table 5-1 lists the suffixes recognized by various Prime software.

INVOKING THE COMPILER

The compiler is invoked from PRIMOS command level by the command:

```
    compiler pathname [options]
```

compiler is the compiler for the language in which the source program is written. Current compilers are:

Compiler	Language
COBOL	COBOL
F77	FORTRAN 77
FTN	FORTRAN IV
Pascal	Pascal
PL1G	PL/I Subset G
RPG	RPG II

Table 5-1. Recognized Filename Suffixes

Suffix	Meaning	Recognized by:	Supplied by:
BASIC	BASIC/VM source file	BASIC/VM	user
BIN	Binary file (created by compiler)	LOAD, SEG	compilers or user
COBOL	COBOL source file	COBOL compiler	user
CPL*	CPL program	RESUME, CPL JOB, PH	user
F77	FORTRAN 77 source file	F77 compiler	user
FTN	FORTRAN IV source file	FTN compiler	user
LIST	Listing file (created by compiler)		compilers, SPSS, user
PASCAL	Pascal source file	PASCAL compiler	user
PL1G	PL/I, Subset G, source file	PL1G compiler	user
PMA	PMA source file	PMA assembler	user
RPG	RPGII source file	RPG compiler	user
SAVE	Runfile created by R-mode loader, LOAD	LOAD, RESUME	LOAD or user
SEG	Segment directory created by SEG	SEG	SEG or user
SPSS	SPSS data file	SPSS	user

* = Required

pathname is the pathname of the source program file. Each compiler recognizes its own suffix. This means that you do not have to specify the suffix when you are invoking the compiler named by the suffix. When a compiler is invoked, it looks first for pathname plus its suffix. If the filename with a 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 then looks for DRAGON.

options allow specification of the creation of object and listing files, the mode in which the object code is to be generated, the types of cross references and listings to be generated, debugger interfaces and the like. These options may be common to all compilers or unique to a particular language. The common options are summarized in Table 5-2 and discussed in the following paragraphs.

OBJECT FILES

In all compilers, the default is to create an object file. The default name of an object file depends on the name of the source file when it is created. If the source file name was created with a compiler identifying suffix, the object file name is filename.BIN. For example, the object filename of PAYROLL.COBOL is PAYROLL.BIN. (For compatability with older files, the default object filename for files without identifying suffixes is B_filename.)

A non-default binary filename can be created (or suppressed) with the -BINARY option (abbreviation -B). This allows you to use the .BIN suffix even if the source filename is not suffixed. Possible arguments for this option are:

Argument	Meaning
-BINARY YES	Create binary file with default name
-BINARY NO	Do not create binary file
-BINARY pathname	Create binary file called pathname

Table 5-2. Compiler Defaults

Compiler	Binary/ Object File	Listing File	Cross-Reference	Mode
COBOL	yes	yes	no	64V
F77	yes	no	no	64V
FTN	yes	no	no	32R
Pascal	yes	no	no	64V
PL1G	yes	no	no	64V
RPG	yes	yes	yes	64R

LISTING FILES

Each compiler can create a file listing the source program. Language-specific options are available to expand on these listings and add more information. The standard listing is generated by default for all compilers except FTN and F77. The option to create a listing file is -LISTING (abbreviation -L). The default name, which is formed in the same way as the default object file name, is filename.LIST (or L_filename, if the source filename has no suffix). The arguments for the -LISTING option are:

Argument	Meaning
-LISTING YES	Create listing file with default name.
-LISTING NO	Do not create listing file.
-LISTING pathname	Create listing file called pathname.
-LISTING TTY	Print listing file at terminal.
-LISTING SPOOL	Print listing file on line printer.

CROSS REFERENCE

Each language has its particular cross reference listing. Each lists the program's variables, tells where they appear in the program, and provides other useful information. Specific details are in each language guide. Cross references are listed by default for RPG only. In other languages, the cross-references listing is generated by using the option -XREF in the command line.

CODE GENERATION

The addressing mode in which object code is to be loaded must be chosen at compilation time. Prime's compilers can generate object code to be loaded in several addressing modes. Table 5-3 shows which types of code can be generated by each compiler.

Table 5-3. Code Generation

	32I	64V	64R	32R
FORTRAN 77 (F77)	✓	✓		
FORTRAN IV (FTN)		✓	✓	✓
Pascal	✓	✓		
PL/I Subset G	✓	✓		
COBOL		✓		
RPG			✓	

In general, 64V mode is the mode of choice. This is the default on all compilers except FORTRAN IV (FTN) and RPG II (RPG). At present, the RPG compiler generates only 64R mode code. To generate 64V mode code in FORTRAN IV, use the 64V option in the command line. For example:

```
FTN GOOD -LISTING YES -64V
```

compiles the program GOOD, producing 64V mode code and creating a listing file, GOOD.LIST.

The FORTRAN 77 (F77), Pascal, and PL/I (PL1G) compilers can also generate 32I mode code. 32I mode code 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 in the command line, as in:

```
F77 CHEERS -32I
```

LOADING

All code generated in 64V or 32I mode is loaded with SEG. (This procedure is often called linking on other systems.) Code generated in 32R or 64R mode is loaded with LOAD. These loaders (or linkers) are summarized in Section 5, and explained in detail in the LOAD and SEG Guide.

COMPILER MESSAGES

If a compilation completes successfully, a message to that effect is printed at the user's terminal (or into the user's COMOUTPUT file, if the compilation is not interactive. See Section 7 for information on COMOUTPUT files.) If compilation is not successful, error and/or warning messages will indicate the offending line and the offense. 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 printed by the F77 and PL1G compilers include explanatory comments. Error messages generated by the FTN, COBOL, and RPG compilers are discussed in those language guides.

COMBINING LANGUAGES IN A PROGRAM

Since all high-level languages are alike at the object code level, and since all use the same calling conventions, programs compiled by the FTN, F77, COBOL, or PL1G compilers can call subroutines compiled by any of the other three compilers. For example, a program written in COBOL could call a subroutine written in FORTRAN 77 which might use a utility subroutine written in PL/I-G. Procedures compiled by the high-level language compilers may also call, or be called by, procedures written in Prime's assembler language, PMA. The following cautions, however, should be observed:

- All I/O routines should be written in a single language.
- Be sure that there is no conflict in data types for variables being passed as arguments. For example, an integer in FORTRAN should be declared as fixed binary in PL/I. Also, remember that PL/I and COBOL may not interpret structures identically.
- All procedures within a program must use compatible addressing modes. Do not put R-mode procedures into a V-mode or I-mode program, or vice versa. (V-mode and I-mode are compatible within programs.)
- Some special restrictions must be observed when FTN and F77 routines are linked together. These are discussed in The FORTRAN 77 Reference Guide.

SECTION 6

LOADING PROGRAMS

INTRODUCTION

PRIMOS has two utilities for loading programs: SEG and LOAD. SEG loads (and runs) V-mode and I-mode programs; LOAD loads R-mode programs. This section explains the basic use of SEG and LOAD for programs written in high-level languages. Language-specific aspects of loading programs are treated in the individual language guides. The loaders are explained in detail in the LOAD and SEG Reference Guide.

SEG

The PRIMOS SEG utility converts object modules (such as those generated by the FTN, F77, COBOL, and PL1G compilers) into segmented runfiles that execute in the 64V addressing mode and take full advantage of the architecture and instruction set of the Prime 350 and up. Segmented runfiles offer the following advantages:

- Much larger programs: up to 256 segments per user program (32 Megabytes)
- Access to V-mode instructions and architecture (Prime 350 and up) for faster execution
- Ability to install shared code: a single copy of a procedure can service many users, significantly reducing paging time
- Re-entrant procedures permitted: procedure and data segments can be kept separate

The following description emphasizes the commands and functions that are of most use to high-level language programmers. Extended features, as well as a complete description of all SEG commands, including those for advanced system-level programming, are described in The LOAD and SEG Reference Guide.

USING SEG UNDER PRIMOS

SEG is invoked by PRIMOS command:

```
    SEG [pathname]
```

or

```
    SEG -LOAD
```

A pathname is given only when an existing SEG runfile is to be executed. (See Section 7.) Otherwise, the command transfers control to SEG command level, which prints a "#" prompt character and awaits a SEG command. After executing a subcommand successfully, the loader repeats the prompt character. (SEG's loader prints a $ prompt to request its subcommands.)

When the -LOAD option is used, SEG enters the loader automatically and requests LOAD subcommands. Using this option also causes SEG to create a segment directory with the filename x.SEG, where x is the base name of the first file loaded. (Details and examples of this loading process are shown later in this chapter.)

If an error occurs during an operation, SEG prints an error message, then the prompt character. Error messages and suggested handling techniques are discussed in this section and in Appendix D.

When a system error (File in use, Illegal name, Insufficient access rights, etc.) is encountered, SEG prints the system error and returns the prompt symbol.

SEG remains in control until a QUIT subcommand returns control to PRIMOS, or an EXECUTE subcommand starts execution of the loaded program.

SEG subcommands can be used in command files, but comment lines are accepted only within its LOAD subprocessor.

NORMAL LOADING

Loading is normally a simple operation with only a few straightforward commands needed. (SEG has many additional features to optimize runfile size or speed, perform difficult loads, load for shared procedures, and deal with possible complications. These are described in The LOAD and SEG Reference Guide.)

The following commands (shown in abbreviated form) accomplish most loading functions.

SEG-Level Commands

DELETE	Deletes segmented runfile.
HELP	Prints a list of SEG commands at terminal.
LOAD	Invokes loader subprocessor for entry of subcommands.

LOAD Subcommands

LOAD pathname	Loads specified object file.
LIBRARY [filename]	Loads library object files from UFD LIB. (Default is PFTNLB and IFTNLB)
MAP [option]	Prints loadmap. Option 3 shows unresolved references (usually subroutines which have not been loaded). Mapping is explained in The LOAD and SEG Reference Guide.
INITIALIZE	Returns loader to starting condition in case of command errors or faulty load.
SAVE	Saves loaded memory image as runfile.
RETURN	Returns to SEG command level.
QUIT	Return to PRIMOS.

Note

SEG recognizes the .BIN suffix for binary files. Thus, to tell SEG to load the file X.BIN, you need only type "LO X". SEG also recognizes the .SEG suffix for subcommands that take segment directories as input files: for example, DELETE and RESTORE. Thus, saying:

```
OK, SEG
# DELETE X
```

deletes X.SEG (if that segment directory exists). If X.SEG doesn't exist, SEG looks for the segment directory X, deleting it if it finds it.

Most loads can be accomplished by the following basic procedure:

1. Give the command SEG -LOAD.

2. Use the LOAD subcommand to load the object file and any separately compiled subroutines.

4. Use the LIBRARY subcommand to load subroutines called from libraries.

5. If you do not receive a LOAD COMPLETE message, do a MAP 3 to identify the unsatisfied references, and load them. If the unsatisfied references are the result of having misspelled some subroutine names, you may want to initialize and re-do the load.

6. When you have gotten the LOAD COMPLETE message, SAVE the runfile. SEG will give the runfile the default name filename.SEG, where filename is the name (without suffix) of the first object file loaded.

After a successful load, you can either start runfile execution from loader command level, or quit from the loader and start execution through the PRIMOS RESUME command. An example of such a load is:

```
OK, SEG -LOAD
$ LO DRAGON
$ LI
LOAD COMPLETE
$ SA
$ QU
OK,
```

If you want to specify your own name for the segment directory, use the following sequence for loading:

1. Invoke SEG from PRIMOS level.

2. Enter the LOAD command to initiate the loading process. At this point, SEG requests a name for the segment directory to be created. Type in the name you desire. (SEG will add the .SEG suffix automatically.)

3. Use the LOAD subcommand to load the object file and any separately compiled subroutines.

4. Use the LIBRARY subcommand to load subroutines called from libraries.

5. If you do not receive a LOAD COMPLETE message, do a MAP 3 to identify the unsatisfied references, and load them. If the unsatisfied references are the result of having misspelled some subroutine names, you may want to initialize and re-do the load.

6. SAVE the runfile.

If these commands produce a LOAD COMPLETE message, then loading was accomplished. If there is a problem, it will become apparent by the absence of a LOAD COMPLETE message or some other SEG error message. (See Appendix D for a complete list of all SEG error messages and their probable cause and correction.)

After a successful load, you can either start runfile execution from loader command level, or quit from the loader and start execution through the PRIMOS SEG command. An example of such a load is:

```
OK, SEG
[SEG REV 18.1]
# LOAD
SAVE FILE TREE NAME: #BENCH9
$ LO B_BENCH9
$ LI
LOAD COMPLETE
$ SA
$ QU

OK,
```

Order of Loading

The following loading order is recommended:

1. Main program

2. Separately compiled user-generated subroutines (preferably in order of frequency of use)

3. Language-specific libraries (PL1GLB for PL/ I, PASLIB for Pascal, VCOBLB and possibly NCOBLB for COBOL)

4. Other Prime Libraries (LI filename), such as VAPPLB(V-mode applications library), VSRTLI (V-mode sort library), VDKALB (MIDAS library)

5. Standard Prime library (LI)

For example, a COBOL program which uses MIDAS files would be loaded as follows:

```
OK, SEG -LOAD
[SEG REV 18.1]
$ LOAD MAIN      Main program first.
$ LOAD SUBR      Separately compiled subroutine next.
$ LI VCOBLB      Shared COBOL library: always used.
$ LI NCOBLB      Non-Shared library: used with separately-compiled
                 subroutines
$ LI VKDALB      MIDAS library: used with MIDAS files.
$ LI             Standard (FORTRAN) library.
LOAD COMPLETE
$ SAVE           Save the file image
$ QUIT           Return to PRIMOS command level.
OK,
```

THE R-MODE LOADER

The PRIMOS LOAD utility converts object modules (such as those generated by the FTN or RPG compilers) into runfiles that execute in the 32R or 64R addressing modes. (Runfiles to execute in the 64V mode must be loaded using the segmentation utility, SEG.)

LOAD recognizes the .BIN suffix for object files. If you give the name X, LOAD looks for X.BIN. If it does not find X.BIN, it looks for X.

LOAD uses the .SAVE suffix for the runfiles it creates.

The following description emphasizes the loader commands and functions that are of most use to the FORTRAN and RPGII programmer. For a complete description of all loader commands, including those for advanced system-level programming, refer to The LOAD and SEG Reference Guide.

USING THE LOADER UNDER PRIMOS

The PRIMOS command:

```
    LOAD
```

transfers control to the R-mode loader, which prints a $ prompt character and awaits a loader subcommand. After executing a command successfully, the loader repeats the $ prompt character.

If an error occurs during an operation, the loader prints an error message, then the $ prompt character. Loader error messages and suggested handling techniques are discussed elsewhere in this section and in Appendix D. Most of the errors encountered are caused by large programs where the user is not making full use of the loader capabilities.

When a system error (File in use, Illegal name, Insufficient access rights, etc.) is encountered, the loader prints this system error and returns its prompt symbol, $.

The loader remains in control until a QUIT or PAUSE subcommand returns control to PRIMOS, or an EXECUTE subcommand starts execution of the loaded program.

Load subcommands can be used in command files, but comment lines result in a CM (command error) message unless they are preceded by '* '.

NORMAL LOADING

Loading is normally a simple operation with only a few straightforward commands needed. The loader also has many additional features to optimize runfile size or speed, perform difficult loads, and deal with possible complications. For details on these, see The LOAD and SEG Reference Guide.

The following commands (shown in abbreviated form) accomplish most loading functions.

PRIMOS-Level Commands

FILMEM	Initializes user space in preparation for load.
LOAD	Invokes loader for entry of subcommands.
RESUME	Starts execution of a loaded, SAVEd runfile.

LOAD Subcommands

DC	Defers loading of COMMON until everything else has been loaded. This prevents overlap of COMMON and program areas.
MODE option	Sets runfile addressing mode as D32R (default) or D64R.
LOAD pathname	Loads specified object file.
LIBRARY [filename]	Loads library object files from UFD LIB. (Default is FTNLIB.)
MAP [option]	Prints loadmap. Option 3 shows unresolved references.
INITIALIZE	Returns loader to starting condition in case of command errors or faulty load.
SAVE [pathname]	Saves loaded memory image as runfile. If pathname is not given, LOAD creates a filename from the name (without suffix) of the first object file loaded plus the .SAVE suffix.
QUIT	Return to PRIMOS.

Most loads can be accomplished by the following basic procedure:

1. Use the PRIMOS FILMEM command to initialize memory.
2. Invoke LOAD.

3. Use the MODE command to set the addressing mode, if necessary. (The default is 32R mode.)

4. Use loader's LOAD subcommand to load the object file and any separately compiled subroutines. (LOAD will search first for filenames plus the .BIN suffix, then for the given filenames without the suffix.)

5. Use loader's LIBRARY subcommand to load subroutines called from libraries (the default is FTNLIB in the UFD LIB). Other libraries, such as SRTLIB or APPLIB, must be named explicitly.

6. If you do not have a LOAD COMPLETE, do a MAP 3 to identify the unsatisfied references, and load them. (If the DC option is used, the LOAD COMPLETE message may not appear until the SAVE command has been given.)

7. SAVE the runfile, either by giving an appropriate name (to which LOAD will add the .SAVE suffix), or by allowing LOAD to create a default filename. (The default is the name (without suffix) of the first object file loaded, plus the .SAVE suffix.)

If these commands produce a LOAD COMPLETE message, then loading was accomplished. If there is a problem, it will become apparent by the absence of a LOAD COMPLETE message or some other loader error message. (See Appendix D for a complete list of all loader error messages and their probable cause and correction.)

After a successful load, you can either start runfile execution from LOAD command level, or quit from the loader and start execution through the PRIMOS RESUME command. An example of such a load is:

```
OK,FILMEM
OK, LOAD
$ DC
$ LO WYVERN
$ LI
LOAD COMPLETE
$ SA
$ QU
OK,
```

Order of Loading

The order of loading, procedures for mapping, etc., are the same for LOAD as they are for SEG.

SECTION 7

RUNNING PROGRAMS INTERACTIVELY

OVERVIEW

Program Environments

Under PRIMOS, programs may execute in three environments:

- Interactive
- Phantom
- Batch

Interactive: This is the environment most often used. In it, program execution is initiated directly by the user. The terminal is dedicated to the program during execution. The program will accept input from the terminal and will print at the terminal any output specified by the program as well as user- or system-generated error messages. Major uses are:

- Program development and debugging
- Programs requiring short execution time
- Data entry programs such as order entry, payroll, etc.
- Interactive programs such as the Editor, etc.

Phantom User: The phantom environment allows programs to be executed while "disconnected" from a terminal. Jobs run as phantoms accept input from a command file instead of a terminal; output directed to a terminal is either ignored or directed to a file.

Major uses of phantoms are:

- Programs requiring long execution time (such as sorts)
- Certain system utilities (such as line printer spooler)
- Freeing terminals for interactive uses

Batch Jobs: Since the number of phantoms on a system is limited, phantoms are not always available. The Batch environment (explained in Section 9) allows users to submit non-interactive command files as Batch jobs at any time. The Batch monitor (itself a phantom) queues these jobs and runs them, up to six at a time, as phantoms become free.

These three environments offer users the following choices for executing programs:

Interactive execution: Users can execute programs directly by using the RESUME or SEG commands, as discussed later in this section. They can also execute programs from either a command procedure file (also called a CPL program) or a command input file (also called a COMINPUT file).

These files allow sequences of PRIMOS commands and subcommands to be written into text files, using the Editor. Invoking the files then executes the commands.

In addition, CPL programs can make run-time decisions about program execution. They can have arguments passed to them, define variables, and execute if statements, loops, and goto's. They are thus much more powerful than COMINPUT files, and are generally preferred.

CPL programs are discussed in Section 8. COMINPUT files are discussed in Section 9.

Phantom execution: Either CPL programs or special phantom files can execute programs in a phantom environment. Phantoms are discussed in Section 9.

Batch execution: Either CPL programs or COMINPUT files can be run as Batch jobs. Batch execution is discussed in detail in Section 10.

What This Section Contains

This section treats the following topics:

- Use of the SEG command to execute segmented runfiles
- Use of the RESUME command to run R-mode runfiles
- Run-time error messages

EXECUTING SEGMENTED RUNFILES

For programs loaded and saved by SEG, execution is performed at the PRIMOS command level using the SEG command:

```
SEG pathname
```

where pathname is the name of a SEG runfile. (SEG looks first for pathname.SEG, then for pathname.) SEG loads the runfile into segmented memory and starts execution. SEG should be used for runfiles created by SEG's loader; it should not be used for program memory images created by the R-mode loader. Example:

```
OK, SEG TEST            /* user requests program
THIS IS A TEST          /* output of program
OK,                     /* PRIMOS requests next command
```

Upon completion of program execution, control returns to PRIMOS command level.

A SEG runfile may be restarted by the command:

```
S 1000
```

if both the SEG runfile and the copy of SEG used to invoke it are in memory.

EXECUTING R-MODE MEMORY IMAGES

For programs loaded in 32R or 64R mode by the loader, execution is performed at the PRIMOS level using the RESUME command. Command line format for RESUME is:

```
RESUME pathname
```

RESUME brings the memory-image program pathname from the disk into the user's memory, loads the initial register settings, and begins execution of the program. For example:

```
OK, R TEST          /* User requests program
THIS IS A TEST      /* Output of program
OK,                 /* PRIMOS requests next command
```

Note

RESUME should not be used for segmented (64V mode) programs. Use the SEG command (discussed in the first part of this section) instead.

When RESUME is invoked, it looks first for pathname.SAVE. If pathname.SAVE cannot be found, RESUME looks for pathname.CPL. If pathname.CPL cannot be found, RESUME looks for pathname without an identifying suffix. For fastest search time, therefore, runfiles should use the .SAVE suffix.

The Start Command

If a program has been made resident in memory (for example, by a previous RESUME command), the START command may be used to initialize the registers and begin execution. Its format is:

```
    START [start-address]
```

If START is typed without a value for start-address, the program resumes at the address value at which execution was interrupted. To restart the program at a different point, specify an octal starting location as the start-address value; the usual default value for the beginning of FORTRAN programs is 1000. For example:

```
    OK, R TEST1        /* Begin
    INPUT NEW KEY: 5   /* Program asks for input
    QUIT               /* User hit BREAK to stop
    OK, S 1000         /* Restart program from beginning
    INPUT NEW KEY:
```

START can also restart a program that has returned control to PRIMOS (for example, because of an error or a FORTRAN PAUSE or CALL EXIT statement).

The applications programmer will almost always use the default forms of the RESUME and START commands (the form discussed here). For a complete treatment of these commands, see The PRIMOS Commands Reference Guide.

Upon completion of the program, control returns to PRIMOS command level.

RUN-TIME ERROR MESSAGES

During program execution, error conditions may be generated and detected by the FORTRAN mathematical functions, file system subroutine calls, or the operating system. A list of run-time error messages is given in Appendix D.

Error messages specific to execution of segmented programs are labeled 64V mode. Some error messages imply system problems beyond the scope of the applications programmer. If so, this is indicated in the explanation of a given error message.

SECTION 8

THE BASICS OF CPL

WHAT IS CPL?

CPL is Prime's "Command Procedure Language" -- a high-level language that operates at PRIMOS command level. CPL provides:

- Variables
- Function calls
- Branching (via such directives as &IF...&THEN...&ELSE, &GOTO, &SELECT)
- Error handling and debugging facilities

LEARNING CPL

CPL provides features both for users who want maximum ease of use for simple programs and for users who want maximum power and flexibility. This section provides a brief overview of CPL, and an introduction to the major "ease of use" features. The CPL User's Guide provides full tutorial and reference information for all CPL features.

HOW DOES CPL WORK?

CPL has two parts: the language and the interpreter. The CPL language allows users to write CPL programs which contain either a sequence of PRIMOS commands or a combination of PRIMOS commands and CPL directives. The commands give instructions to PRIMOS, or to one of its subsystems. The directives give instructions to the CPL interpreter itself. (PRIMOS never sees these directives; it sees only the commands which the interpreter passes to it.)

When the programs are executed, 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. (Figure 8-1 illustrates the evaluation of a CPL directive and the resulting execution of a command.) Thus, a lengthy series of commands can be set in motion by a single command, relieving the user of much repetitive typing; yet run-time decisions can be made at any time during the file's execution.

1. CPL file contains the statement:	IF %A%>%B% THEN F77 %FILENAME%
2. The CPL interpreter reads the statement, substituting current values for variable references:	IF 3>1 THEN F77 JEFF
3. The CPL interpreter tests:	3>1? TRUE
4. Since the test condition is true, CPL executes the THEN statement, passing the command "F77 JEFF" to the Standard Command Processor:	THEN F77 JEFF
5. Command processor executes the command:	F77 JEFF

Figure 8-1. Execution of a Sample CPL Directive

CREATING AND EXECUTING CPL PROGRAMS

Like other high-level language programs, CPL programs are written using the editor (ED). Their format is simple, being based on the principle of one statement per line. Indentation may be used as desired, for ease of reading. (Format rules are explained where applicable throughout this section. They are explained fully in Section 3 of The CPL User's Guide.)

CPL programs must be given names that end with the .CPL suffix.

CPL programs are not compiled or loaded. As soon as they have been written, they are ready to execute.

You may run CPL programs interactively with either the CPL command or the RESUME command. You may also run them as phantoms (with the PHANTOM command), or as Batch jobs (with the JOB command). For details of how to run CPL programs as Batch jobs, see Section 10.

You do not need to specify the .CPL suffix when you submit a CPL program for execution. The CPL, RESUME, JOB, and PHANTOM commands will all look for filename.CPL when you specify filename. Their search rules are as follows:

- The CPL command looks first for filename.CPL, then for filename. It runs either one as a CPL program.
- The RESUME command looks first for filename.SAVE, then for filename.CPL, then for filename. It runs files whose names end in CPL as CPL programs. It runs all other files as run-time (compiled and loaded) programs.
- The PHANTOM and JOB commands look first for filename.CPL, then for filename. They run files whose names end in .CPL as CPL programs; they run other files as command input files.

DEBUGGING CPL PROGRAMS

Syntax Error

If syntax errors prevent a CPL program from executing, the interpreter prints a substantial amount of information at the user's terminal and/or into a COMOUTPUT file. (For details on COMOUTPUT files, see Section 9.) The information 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. If the error-causing text can be printed, it will be part of the 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, program BLUNDER.CPL contained a misprint, FILLNAME, for the variable, FILENAME.

If CPL programs are halted by PRIMOS errors, then the normal PRIMOS error message (ending in "ER!") is printed out.

Note

Either CPL or PRIMOS syntax errors halt the CPL program and return the user to command level. However, warning messages from PRIMOS or its subsystems will not normally halt execution of a CPL program.

CPL's &SEVERITY directive (explained in The CPL User's Guide) allows users to modify a program's response to PRIMOS errors and warnings.

Logic Errors

If a CPL program runs, but produces erroneous 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 print out the value of a variable each time the value is set or altered.
- echoing, to print commands and directives as they are read. (This can tell you if unexpected branching is occurring.)
- 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, see Section 10 of The CPL User's Guide.

USING PRIMOS COMMANDS IN CPL PROGRAMS

The simplest CPL programs are composed entirely of PRIMOS commands. For example, a CPL file might execute three programs. Such a program might be named RUNNEM.CPL. It might look like this:

```
RESUME NEW_TALLY
RESUME NEW_SORT
SEG DAILY_UPDATE
```

Which PRIMOS Commands Can you Use?

CPL files that consist entirely of PRIMOS commands can use the following commands:

- All compiler commands: COBOL, F77, FTN, PL1G, PMA, RPG, etc.
- All commands which execute programs. For example:

  ```
  SEG THISFILE.SEG
  R THATFILE.SAVE
  R FILE.CPL
  BASICV ANYFILE
  ```

- Any user commands which do not invoke a subsystem or initiate a dialog. For example, you may use:

  ```
  ATTACH
  LISTF
  CREATE
  DELETE
  CNAME
  PASSWD
  PROTEC
  SIZE
  ```

- Commands that invoke interactive subsystems or user programs, if the user is going to supply the data or subcommands from the terminal at runtime. For example:

  ```
  ED
  SEG
  MAGNET
  SORT
  ```

If you want the CPL program itself to supply the data or subcommands, you must use CPL's &DATA directive, explained later in this section.

What Commands Can't You Use?

Do not use the commands

- COMINPUT (in any form)
- CLOSE ALL
- DELSEG ALL

in a CPL file. Any of these commands will abort execution of the file.

CPL DIRECTIVES

The CPL langauge contains the following directives. Those marked with asterisks are discussed in the remainder of this section. All are discussed in detail in The CPL User's Guide.

Directive	Use
VARIABLE- AND ARGUMENT-HANDLING DIRECTIVES	
* &ARGS	Defines names (plus types and default values, if desired) for 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	
* &IF...&THEN...&ELSE	Allows conditional branching, choosing between Boolean (TRUE-FALSE) alternatives.
&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 "wild card" capabilities.
* &DATA...&END	Groups statements to be treated as data or subcommands for user programs or PRIMOS utilities (such as ED or FUTIL).
&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. The &RETURN directive can also pass messages and/or integer severity codes to the user or caller of the halted program or routine.
&STOP	Halts execution of a CPL program, whether it is used in the main program or in an internal routine. The &STOP directive can also pass messages and/or integer severity codes to its program's caller.

SUBROUTINES AND USER-DEFINED FUNCTIONS
(INTERNAL AND EXTERNAL PROCEDURES)

&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

&DEBUG	Turns on (or off) CPL's debugging facility during program execution. Options to these directives specify debugging actions to be taken.
&EXPAND	Allows use of specified ABBREV file by the CPL program. (For details on ABBREV files, see Section 15.)

&SEVERITY	Defines the behavior of the CPL program (stop, continue, or call an error-handling routine) when system-defined errors or warnings occur.

ERROR- AND CONDITION-HANDLING DIRECTIVES

&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. (See Section 16 for information on conditions and on PRIMOS's Condition Mechanism.)
&REVERT	Disables a specified condition handler.
&SIGNAL	Signals a user-defined (or system-defined) condition to the condition mechanism.

USING VARIABLES IN CPL PROGRAMS

There are three ways in which CPL programs can obtain variable data:

- The variable data can be passed to the CPL programs as arguments when the CPL program is invoked. In this case, the &ARGS directive is used inside the CPL program to define variable-names for the arguments and to match these names to the data supplied in the invocation.

- Variables may be defined and assigned values inside the CPL program by using the &SET_VAR directive.

- A user can maintain a global variable file. (See Section 15 for details.) By using the command "DEFINE_GVAR filename" in a CPL program, the user allows the program to access that file and reference the variables contained in it.

Once a variable has been defined in a CPL program (by the &ARGS or &SET_VAR directives, or by the DEFINE_GVAR command), it may be referenced by placing its name inside percent signs. Thus, if NAME was the name of the variable, %NAME% would be a reference to that variable. When the CPL interpreter read the reference, it would substitute the current value of NAME for the string %NAME% in the command line or directive in which the reference occurred.

Using the &ARGS Directive

The simplest form of the &ARGS directive is:

```
&ARGS variable_name [;...variable_name]
```

For example, a CPL program (named F7.CPL) that compiles any F77 source file might read:

```
&ARGS FILENAME
COMO  %FILENAME%.COMO
DATE
F77   %FILENAME% -DEBUG
COMO -E
```

In this example, the &ARGS directive defines one variable, FILENAME. When the file is invoked, the name of the file to be compiled is supplied as an argument, following the name of the CPL file. For example, the invocation might read:

```
R F7 JEFF
```

In this example, the &ARGS directive takes the character string JEFF and assigns it to the variable FILENAME. JEFF is now the value of FILENAME.

From now on, each time a variable reference, %FILENAME%, is found, the CPL interpreter substitutes the character string JEFF for the character string %FILENAME%. Thus, the command:

```
COMO %FILENAME%.COMO
```

becomes

```
COMO JEFF.COMO,
```

while the command

```
F77 %FILENAME% -DEBUG
```

becomes

```
F77 JEFF -DEBUG.
```

Note that the variable, FILENAME, is not enclosed in percent signs when it is being defined in the &ARGS directive, but is enclosed in percent signs whenever it is "referenced"--that is, whenever its value, rather than its name, is wanted.

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 juxtaposed, (as in %FILENAME%%FILENAME%). Again, a single string results (JEFFJEFF).

Multiple Arguments

When multiple arguments are given, the variable names in the &ARGS directive must be separated by semicolons. For example:

```
&ARGS FILENAME;  COMPILER
```

Now you can write a more general CPL file, called COMPILE_ALL.CPL, that can compile FTN, F77, or PL1G source files. It reads:

```
&ARGS FILENAME; COMPILER
COMO  %FILENAME%.COMO
DATE
%COMPILER% %FILENAME% -64V -DEBUG
COMO -E
```

Invoking this file by typing:

```
R COMPILE_ALL JEFF FTN
```

creates the command,

```
FTN JEFF -64V -DEBUG
```

In general, arguments are defined by their position in the command line. In the above example, the first argument, "JEFF", became the value of the first variable in the &ARGS line, "FILENAME". The second argument, "FTN", was assigned to the second variable, "COMPILER". Giving the arguments in reverse order:

```
R COMPILE_ALL FTN JEFF
```

would assign "FTN" to "FILENAME" and "JEFF" to "COMPILER".

Omitted Arguments

If an argument is omitted from the command line, the CPL interpreter sets its value to the explicit null string, ''. The PRIMOS command processor then removes the null string before executing the command. In the above example, the command:

```
R COMPILE_ALL TESTFILE
```

assigns the value TESTFILE to the variable FILENAME, and assigns the null string to the variable COMPILER. The resulting PRIMOS command first becomes:

```
'' TESTFILE -64V -DEBUG
```

and then becomes:

```
TESTFILE -64V -DEBUG
```

Since PRIMOS can do nothing worthwhile with such a command, it returns you to command level with no compilation having occurred.

<u>Note</u>

> CPL offers several ways to deal with null arguments. These are explained in <u>The CPL User's Guide</u>.

<u>The &SET VAR Directive</u>

The form 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.

<u>value</u> 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 the value of 10, Y the value of 5, and Z the value of 15.

<u>Note</u>

> In CPL programs, all operators MUST be separated from their operands by one or more spaces.

DECISION-MAKING (BRANCHING) IN CPL PROGRAMS

When a CPL program contains only PRIMOS commands (or PRIMOS commands plus variables), it is executed sequentially; that is, each command (each line of the program) is executed in turn.

Sometimes, however, you may want to alter the sequence in which the commands are executed. To alter the "flow of control" in this way, you use CPL's flow of control directives. The simplest and most important of these is the &IF directive.

The &IF Directive

The form of the &IF directive is:

&IF test &THEN statement

Test is a logical test which can be answered TRUE or FALSE (for example, &IF A = B, &IF %NUMBER% < 10). Statement is either a command or a CPL directive.

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)

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 if test is true or false. If test is true, the interpreter executes the command or directive that forms the &THEN statement.

An example: Suppose you compile a program frequently, but only occasionally want to spool the listing file. You could use an argument and the &IF directive to tell the CPL program whether or not to spool the listing file. Here's a program to do it (called CNS.CPL):

Note

As this program shows, you can use /* to place comments in CPL programs.

```
&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

```
    R CNS JEFF SP
```

then the test, SP = SP, is true, and the listing file, JEFF.LIST, is spooled. If you give the command

```
    R 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 example 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. (In the example, spool the file, or don't spool it.) When the &IF and &ELSE directives are used together, they tell the interpreter to choose between two courses of action.

The form 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 might want a program (called COMPILE2.CPL) that looked like this:

```
    &ARGS FILENAME; COMPILER
    &IF %COMPILER% = F77 &THEN F77 %FILENAME% -DEBUG -32I
    &ELSE FTN %FILENAME% -64V
```

If you give the command "R COMPILE2 THISFILE F77", the test (F77 = F77) becomes true, and THISFILE is compiled by the F77 compiler. If you give any other value for the "compiler" argument--or if you omit that argument altogether--THISFILE is compiled by the FTN compiler.

Nested &IFs

&IF directives may be nested: that is, 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 the command groups.

&DO Groups

The format for &DO groups is as follows:

```
&DO
    statement 1
    statement 2
    .
    .
    .
    statement n
    &END
```

Normally, each statement in a CPL program represents one action the interpreter is asked to perform. In a &DO group, however, all the statements between the &DO and the &END represent a single action to the interpreter. Thus instead of saying

```
&IF test &THEN statement-1
     &ELSE statement-2
```

we can say

```
&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
     SEG MONTHLY_REPORT
     SEG END_OF_YEAR_REPORT
     SEG XMAS_LIST
     &END
&ELSE SEG 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; that is, functions and their arguments 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 character string thus produced for the character string represented by the function call.

The NULL function: One of the most useful CPL functions is the NULL function. Its form is

```
[NULL var]
```

where var is any CPL variable.

The NULL function tests for a null character string, returning the character string TRUE if it finds one and the character string FALSE if it does not. Since 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.

An example: A test for a null argument might be used to set the home UFD for some procedure. For example, a CPL program might begin

```
&ARGS WHERE
IF [NULL %WHERE%] &THEN ATTACH MY_UFD
     &ELSE ATTACH %WHERE%
```

Specifying WHERE allows you to make any desired ATTACH; omitting WHERE attaches you to your default choice (MY_UFD).

The EXISTS Function

The EXISTS function is a Boolean function that determines

- Whether or not a file system object exists
- Whether it matches a specified type (file, directory, or segment directory)

The form of the function call is:

```
[EXISTS pathname [type]]
```

pathname is the name or pathname of a file or directory.

type is one of the following:

```
-ANY
-FILE
-DIRECTORY or -DIR
-SEGMENT_DIRECTORY or -SEGDIR
```

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 on the existence or non-existence of pathname.

Examples: The first example checks to see if a "new" file has been written. If it has, it calls ED to allow its user to edit the new file. If the new file does not exist, the program requests the older version:

```
&IF [EXISTS MEMO.NEW]  &THEN ED MEMO.NEW
    &ELSE ED MEMO
```

The second example uses the "NOT" symbol, ^, to reverse the value returned by EXISTS. This program wants to attach to a specific directory. If the directory doesn't exist, it will create it before doing the ATTACH:

```
&IF ^ [EXISTS SUBDIR] &THEN CREATE SUBDIR
ATTACH *>SUBDIR
```

USING CPL WITH SUBSYSTEMS: &DATA GROUPS

Many of Prime's utilities, such as ED (the text editor) and SEG (the V-mode and I-mode loader), require subcommands to accomplish their function. Similarly, many user programs require that data be typed in from 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 (&DO, &DATA), 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".

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 that compiles, loads, and executes a PL/I-G program:

```
/*CPL program to compile, load, and execute a PL1G program
/*usage:  R CLR FILENAME
/*
PL1G %FILENAME%           /*Compile program
/*
&DATA SEG -LOAD           /*Invoke SEG
  LOAD %FILENAME%         /*Provide SEG commands
  LI PL1GLB               /*via &data directives
  LI
  SA
  QU
&END                      /*end of &data group
SEG %FILENAME%.SEG        /*execute run-file
```

Terminal Input in &DATA Groups

Sometimes 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 at the end of the &DATA group, just before the &END statement.

The format is:

```
&DATA
 statement-1
 .
 .
 .
 statement-n
&TTY
&END
```

When execution reaches the &TTY directive, control returns to the user at the terminal. When the user leaves the subsystem, control returns to the CPL file.

An Example

This example shows how the &TTY directive might work with a user program. Assume 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:
```

A given department (for instance, the hardware department) might use a CPL program (named P.CPL) to invoke the PURCHASE program and pass it its first two items of information. The statements would look like this:

```
&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 terminal session might look like this:

```
OK, R P
dept. name: HDWR
dept. number: 38
customer name: H.L. Smith
acct. number: 35684
amount of purchase:  536.89
OK,
```

Note

By using a loop and the RESPONSE function, you could write a CPL program that would pass information for any number of purchases to program PURCHASE. The *CPL User's Guide* explains how to do this.

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 may 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
```

SECTION 9

COMMAND FILES AND PHANTOMS

INTRODUCTION

This section discusses:

- How to create and run COMINPUT files
- How to create COMOUTPUT files
- How to use DATE, TIME, and RDY in command files
- How to run phantoms

Batch execution of command files will be discussed in Section 10.

COMMAND FILE REQUIREMENTS

Command input files may contain any legal PRIMOS commands, utility subcommands, or dialog responses, on a line-for-line basis (i.e., each line in the file must correspond to a line as it would be typed at a terminal.) Each utility except Batch imposes certain requirements:

- For COMINPUT, the last command should be COMINPUT -TTY or COMINPUT -END.
- For PHANTOM, the last command should be LOGOUT.
- Any command file can be used for Batch.

Comments

Command input files can be made self-documenting by including comment lines at PRIMOS command level. A line beginning with a slash and asterisk, (/*), is interpreted as a comment and is ignored by PRIMOS. If a command output file is open, any comments entered at the terminal by the user or from a command file are written into the command output file. Any character may be used in a comment line. A comment may also be appended to a command at PRIMOS command level as in:

```
    SLIST BENCH07.MAP            /* PRINT MAP FILE
```

THE COMINPUT COMMAND

The COMINPUT command causes PRIMOS to read input from a specified command file rather than from the terminal. Commands are executed as if they were entered at the terminal. The format is:

```
COMINPUT [command-file] [-options] [file-unit]
```

command-file	The pathname of the file from which input is to be read.
options	Specify command control flow as detailed below.
file-unit	The PRIMOS file unit number on which the input file is to be opened. If omitted, file unit 6 is used. File units must be octal (i.e., decimal 8 is entered as 10).

Options

-TTY -END	Either one switches the command input stream to the user terminal and closes the command input file.
-PAUSE	Switches command input stream to the user terminal but does not close the command input file.
-CONTINUE	Returns control to command input file following a CO -PAUSE or an error.
-START	Resumes command following a BREAK interruption of execution of a command file.

The -TTY, -END and -PAUSE options are used only within command files. The -CONTINUE and -START options are typed by the user.

The -TTY or -END option must be the final command in the command file (or in the last command file, if files are chained as described, below).

A simple command file, TEST.CO, might be created to compile the program TEST.FTN:

```
/*BEGIN TEST OF COMMAND FILE
COMOUTPUT TEST.COMO
DATE
/*COMPILE THE PROGRAM IN 64V MODE
FTN TEST -64V
/*LOAD THE PROGRAM
SEG -LOAD
LO TEST
LI
SA
```

```
    MAP LOADTEST.MAP 7
    MAP UNSATISFIED.MAP 3
    QU
    /*COMMAND FILE TEST COMPLETED
    DATE
    COMO -END
```

The command file would be executed by the command:

```
    CO TEST.CO
```

and would produce the following output file:

```
    OK, DATE

    Thursday, November 20, 1980   4:04 PM

    OK, /*COMPILE THE PROGRAM IN 64V MODE
    FTN TEST -64V
    0000 ERRORS [<.MAIN.>FTN-REV18.1]

    OK, /*LOAD THE PROGRAM
    SEG -LOAD
    [SEG rev 18.1]
    $ LO TEST
    $ LI
    LOAD COMPLETE
    $ SA
    $ MAP LOADTEST.MAP 7
    $ MAP UNSATISFIED.MAP 3
    $ QU

    OK, /*COMMAND FILE TEST COMPLETED
    DATE

    Thursday, November 20, 1980   4:05 PM

    OK, COMO -END
```

Chaining Command Files

The -CONTINUE option of COMINPUT allows command files to be chained. The following example illustrates the chaining of three command files, and shows how file unit conflicts can be avoided. The command file GO.CO contains the following commands:

```
   /* COMPILE THE PROGRAM IN 64V MODE
   FTN TEST -64V
   /* LOAD THE PROGRAM
   COMINPUT LOADTEST.CO 7
   CLOSE 7
```

```
/* RETURN COMMAND TO USER TERMINAL
COMINPUT -TTY
```

The command file LOADTEST.CO contains the following commands:

```
/* LOADTEST COMMAND FILE
SEG -LOAD
LO TEST
LI
SA
QU
COMINPUT MAPS.CO 1Ø
CLOSE 1Ø
COMINPUT -CONTINUE
```

The command file MAPS.CO contains the fc 'owinq mmands:

```
/* GET FULL MAP AND UNSATISFIED REFERENCES
SEG
VLOAD * TEST
MAP LOADTEST.MAP 7
MAP UNSATISFIED.MAP 3
QU
/* RETURN TO 'CALLING' COMMAND FILE
COMINPUT -CONTINUE 7
```

Typing COMINPUT GO.CO causes PRIMOS to read and execute the commands in GO.CO. When the command COMINPUT LOADTEST.CO 7 is reached, control passes to LOADTEST.CO, which loads the object file, then calls MAPS.CO (on file unit '1Ø) to obtain two load maps. When the command COMINPUT -CONTINUE is reached in MAPS.CO, control returns to the statement following the call in LOADTEST.CO, which closes the file unit used for MAPS.CO. When COMINPUT -CONTINUE is reached in LOADTEST.CO, control similarly returns to GO.CO. Finally, the command COMINPUT -TTY in GO.CO returns control to the user's terminal.

```
OK, CO GO.CO
OK, /*COMPILE THE PROGRAM IN 64V MODE
FTN TEST -64V
ØØØØ ERRORS [<.MAIN.>FTN-REV18.1]

OK, /*LOAD THE PROGRAM
COMINPUT LOADTEST.CO 7
OK, /*LOADTEST COMMAND FILE
SEG -LOAD
[SEG rev 18.1]
$ LO TEST
$ LI
LOAD COMPLETE
$ SA
$ QU
```

```
OK, COMINPUT MAPS.CO 10
OK, /*GET FULL MAP AND UNSATISFIED REFERENCES
SEG
[SEG rev 18.1]
#VLOAD * TEST
$ MAP LOADTEST.MAP 7
$ MAP UNSATISFIED.MAP 3
$ QU

OK, /*RETURN TO 'CALLING' COMMAND FILE
COMINPUT -CONTINUE 7
OK, CLOSE 10
OK, COMINPUT -CONTINUE
OK, CLOSE 7
OK, /*RETURN COMMAND TO USER TERMINAL
COMINPUT -TTY
OK,
```

Errors

Non-recoverable errors return input control to the terminal, leaving the command file open. The user may type a correct version of the offending line, and then resume input from the command file by the command CO -CONTINUE.

Closing Command Input Files

In chaining command files, the 'called' files should be closed upon returning to the 'calling' files, either by file unit number (as in the example above) or by filename. The user should make certain that the file units to be used for the command input files are not already opened (or going to be opened) by user programs, utilities, or other command input files.

Note

The CLOSE ALL command should not be used in a command input file, as it closes all files, including the command input file from which this command is read. The message "Unit not open. Cominput (Input from terminal.)" will be printed and input control will be switched to the terminal.

THE COMOUTPUT COMMAND

The COMOUTPUT command writes, into a specified file, both the output stream directed to the terminal by PRIMOS and the input presented to PRIMOS. The input may originate as direct typing, or come from a command file running under COMINPUT, PHANTOM or Batch. The resulting output file is a permanent record of the entire dialog.

Output to the terminal can be suppressed. Print suppression increases speed since it normally takes more time to write to a terminal than to a disk file.

The command format is:

```
   COMOUTPUT [output-file] [-options]
```

output-file is the pathname of the file to which the output stream is sent. options specify terminal and file output and control flow as described below.

Terminal Options

These can be used when the output file is first opened, or at any time before the command output file is closed. User input is always echoed at the terminal even if the -NTTY option is used.

-NTTY	Turn off terminal output.
-TTY	Turn on terminal output (default).

Error messages are printed in the output file and at the terminal, regardless of the terminal option selected. Any inter-user terminal output (e.g., messages from the supervisor terminal) is printed at the terminal but not in the output file.

File Options

These stop or restart output to the command file. They may also be used to append output to an existing file.

-PAUSE	Stop output to command file; leave file open.
-CONTINUE	Resume output (halted by -PAUSE) to the command output file. Or, if at PRIMOS level, re-open an existing COMOUTPUT file and position the pointer so that new output will be appended.
-END	Stop output to command file; close file.

A BREAK turns terminal output on, but does not close the file. A LOGOUT turns terminal output on and also closes the command output file, as well as any other files the user has currently open. For example:

```
   COMO FTNTEST.COMO
```

opens the file FTNTEST.COMO for output and positions the pointer to the start of the file. If FTNTEST.COMO already exists, its previous contents will be deleted immediately. To open an existing file for appending, type:

```
   COMO FTNTEST.COMO -C
```

This opens the file FTNTEST.COMO and positions the pointer at the end of the file.

Closing Command Output Files

Command output files are closed by the COMO -END command. For example:

```
    COMO TEST.COMO
    SLIST RECORDS
    COMO -END
```

USING DATE AND TIME IN COMMAND FILES

The DATE Command

The command DATE prints the system date and time at the user terminal.

```
    OK, DATE
    GO

    Wednesday, October 10, 1979   10:11 AM

    OK,
```

This feature allows command output files to be stamped with date/time information for identification, as an aid to program development and debugging. For example, the sequence of commands:

```
   COMO TEST1.COMO
   DATE
     .
     .
     .
   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 this interactive session.

DATE may also be included in command input files or in command files for Batch execution.

The TIME Command

The command TIME entered at the user terminal prints the current values in the time accounting registers. These are: connect time, compute time, and disk I/O time.

```
    OK, TIME
      1'32   Ø'11   Ø'Ø8
    OK,
```

Connect time is the time since LOGIN (in hours and minutes). Compute time is the time accumulated executing commands or using programs (in minutes and seconds). This does not include disk I/O time. Disk I/O time (in minutes and seconds) is the accumulated time for disk input and output. Disk I/O includes paging I/O time generated on the user's behalf. All times include system supervisor overhead caused by user requirements.

The TIME command can be given before and after executing a program. The time differences can be used to benchmark the program and measure efficiency as the program is optimized.

Example: The command input file BENCHØ7.CO contains the following:

```
    COMO BENCHØ7.COMO
    /* TIMING TEST OF BENCHØ7 PROGRAM
    DATE
    /* GET START TIME VALUES
    TIME
    SEG TEST
    /* GET STOP TIME VALUES
    TIME
    COMO -END
    CO -TTY
```

The command CO BENCHØ7.CO executes this command file. Upon completion, the output file BENCHØ7.COMO contains the following:

```
    OK, /*TIMING TEST OF BENCHØ7 PROGRAM
    DATE

    Thursday, November 2Ø, 198Ø   1Ø:24 AM

    OK, /*GET START TIME VALUES
    TIME
      Ø'11   Ø'Ø3   Ø'Ø3
    OK, SEG TEST
    The answer is  7Ø

    OK, /* GET STOP TIME VALUES
    TIME
      Ø'11   Ø'Ø3   Ø'Ø4
    COMO -END
```

The RDY -LONG Command

An alternate method of measuring program efficiency is provided by the RDY -LONG command. When this command is given, each OK prompt includes the time of day, the amount of CPU time (in seconds) and the amount of I/O time (also in seconds) used since the last prompt.

```
OK, RDY -LONG
OK 09:21:29  0.284  0.324
```

To return prompts to their normal form, use the command RDY -BRIEF:

```
OK 09:21:43  0.036  0.000
RDY -BRIEF
OK,
```

As an example of using the RDY command, let us modify the command file BENCH07.CO:

```
COMO BENCH07A.COMO
/*TIMING TEST OF BENCH07 PROGRAM
DATE
/*use rdy-long for time between prompts
RDY -LONG
SEG TEST
COMO -END
CO -TTY
```

The output file for the new command file is as follows:

```
TIMING TEST OF BENCH07 PROGRAM
DATE

Wednesday, November 19, 1980   10:06 AM

OK, /*use rdy-long for time between prompts
RDY -LONG
OK 10:06:15  3.560  3.924
SEG TEST
The answer is  70

OK 10:06:18  0.287  1.051
COMO -END
```

PHANTOM USERS

The phantom user feature allows command file processing without tying up a terminal. Once a phantom process has been initiated, it is treated by PRIMOS as a separate process that is not associated with a terminal. The terminal is then made available for other uses.

The command file or CPL program run by the phantom process specifies the commands and their sequence, program invocations and necessary input data required to complete a particular job. Phantoms are used for long compilations, loadings, and executions that are debugged and require no interactive terminal input. Certain PRIMOS system utilities (e.g., FAM, SPOOL) are implemented as phantom processes.

Using Phantoms

A phantom user process is initiated by the command:

```
                          {CPL-args }
   PHANTOM  filename      {file-unit}
```

filename is the name (or pathname) of a CPL program or command input file.

If a COMINPUT file or special Phantom command file is to be run, then file-unit may be the PRIMOS file unit number on which the command file is to be opened. If omitted, file unit 6 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 PHANTOM command checks for available phantom processes. The number varies with each installation. The message:

```
   No phantoms are available.  FILENAME
```

is returned if no processes are available. Control is then returned to PRIMOS. When a phantom process is available, the message:

```
   PHANTOM is user user-number
```

is returned and the phantom user is logged in (under the same login-name as the invoker). user-number is the number assigned by PRIMOS to the phantom process. Control returns to PRIMOS, the terminal is freed for other use, and the phantom command file is opened on the specified (or default) unit. PRIMOS then reads all further commands for the phantom user from the command file.

Phantom Operation

Phantom processes should not execute programs which require input from an actual terminal. Such an instruction will abort and log out the phantom process.

While a phantom process is in operation, terminal output is suppressed unless a command output file has been opened by a COMOUTPUT command in the phantom command file. Output is then written to the COMOUTPUT file.

It is possible to initiate another phantom from a running phantom, in a manner similar to chained COMINPUT files. However, there is no guarantee that a phantom user process will be available when the process is requested by a command file.

The final command in the last executed phantom command file should be LOGOUT. If it is not, the phantom will report an abnormal termination when it is logged out.

Phantom Logout

At the completion of a job process, phantom users are automatically logged out. To cancel a phantom user process before completion, use the command:

```
   LOGOUT -user-number
```

user-number is the PRIMOS-assigned phantom user number.

Any phantom can be logged out from the supervisor terminal. From a user terminal, a phantom can be logged out only if the terminal has the same login UFD as that which initiated the phantom.

Logout Notification

When a phantom logs out, notification is sent to the terminal of the user who started the phantom. Normal logout is shown by a message such as:

```
    PHANTOM 87 NORMAL LOGOUT AT 11:27
    time used= 0:1 0:0 0:0
```

Forced logout (the result of an error that halted the phantom program, a deliberate LOGOUT command, or the absence of LOGOUT as the final command in the phantom's final command file) results in a message such as:

```
    PHANTOM 86 ABNORMAL LOGOUT AT 11:13
    time used= 0:1 0:0 0:0
```

In these messages, the figures following the phrase "time used" indicate elapsed time, CPU time, and I/O time used by the phantom process.

If the user who started the phantom logs out before the phantom completes its job, logout notification cannot be sent to the user's terminal. It is possible, however, for users to set up programs to record phantom logout notifications. This is done using the subroutines LO$R and LO$CN. For information on these subroutines, see The PRIMOS Subroutines Reference Guide.

Phantom STATUS Information

The STATUS USER command (discussed in Section 3) provides a list of all the users in the system, their login numbers, assigned line numbers, etc. Phantom users are distinguished by the code PH in the line number field of a STATUS list. For example:

```
OK, status users

USER     NO LINE DISKS
SYSTEM    1  ASR <SYS.K>   AL57
CROW      7    5           (TO NJE    )
PERCH     8    6 <PLAINS>
ELM      11   11           (TO NJB    )
BALSA    13   13           (TO NJB    )
OWL      19   21 <QAGRP3>
HAWK     21   23 <QAGRP3>
CORAL    38   44           (TO NJE    )
WILLOW   46   54 <QAGRP3>
BEECH    49  REM <FOREST>  (FROM NJB    )
PARROT   50  REM <QAGRP3>  (FROM NJE    )
BIRCH    51  REM <PLAINS>  (FROM NJB    )
FAM      94   PH <QAGRP3> <SYS.K>
SYSTEM   95   PH <SYS.K> (2)

OK,
```

Example of Phantom Command File

The phantom command file TEST.PH contains the following commands:

```
/*BEGIN TEST OF PHANTOM
COMOUTPUT TEST.COMO
DATE
/*COMPILE THE PROGRAM IN 64V MODE
FTN TEST -64V
/*LOAD THE PROGRAM
SEG -LOAD
LO TEST
LI
SA
```

```
    MAP LOADTEST.MAP 7
    MAP UNSATISFIED.MAP 3
    QU
    /*PHANTOM TEST COMPLETED
    DATE
    /*COMO -E would normally go here.
    /* It has been omitted so the logout sequence
    /* could be shown in the comoutput file.
    LOGOUT
```

When a phantom is invoked at the terminal by PH TEST.PH, the terminal interactive dialog is:

```
    OK, PH TEST.PH
    PHANTOM is user 61
    OK,
```

The contents of the command file, TEST.COMO, created by the phantom are:

```
    OK, DATE

    Friday, November 21, 1980   10:06 AM

    OK, /*COMPILE THE PROGRAM IN 64V MODE
    FTN TEST -64V
    0000 ERRORS [<.MAIN.>FTN-REV18.1]

    OK, /*LOAD THE PROGRAM
    SEG -LOAD
    [SEG rev 18.1]
    $ LO TEST
    $ LI
    LOAD COMPLETE
    $ SA
    $ MAP LOADTEST.MAP 7
    $ MAP UNSATISFIED.MAP 3
    $ QU

    OK, /*PHANTOM TEST COMPLETED
    DATE

    Friday, November 21, 1980   10:06 AM

    OK, /*COMO -E would normally go here.
    /* It has been omitted so the logout sequence
    /* could be shown in the comoutput file.
    LOGOUT
    BEECH (62) LOGGED OUT AT  10:06 112180
    TIME USED=  0'00   0'01   0'02
```

SECTION 10

BATCH JOB PROCESSING

INTRODUCTION

Batch is the most flexible of the PRIMOS job processing utilities. Any CPL program or command file that will run under PRIMOS can be run as a Batch job. This means that users may write CPL programs for submission as Batch jobs without including special Batch commands. Yet users may also run existing COMINPUT, PHANTOM, and CX files as Batch jobs; Batch will accept them all.

Batch offers further flexibility in job scheduling and execution control. Each Batch queue has a phantom from which to run users' jobs. These phantoms run "in the background" of the system: that is, they run concurrently with interactive jobs, but at somewhat lower priorities. Thus, they use only small amounts of CPU time when interactive use is heavy, but utilize large amounts of CPU time when interactive use is light or absent. 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, can be set up as Batch jobs during the day, then run under operator control at night.

Each Batch queue is a separate entity, defined by the System Administrator to be particularly hospitable to certain types of jobs. Queues designed for short jobs have a fairly high scheduler priority, but a short timeslice; queues designed for normal jobs have slightly lower priorities and normal timeslices. Queues designed for long jobs have low priorities but large timeslices. The queues for short jobs will thus run fastest, as they can operate during times of heavier interactive use. The other queues will take fuller advantage of periods of lighter activity. By using the BATGEN (BATch GENeration) command, explained below, users can see what queues are available and what their characteristics are. They can then submit their jobs to the appropriate queues.

USING THE BATCH SUBSYSTEM

Users communicate with the Batch subsystem through four commands: BATCH, BATGEN, JOB, and $$ JOB. With these commands, they can:

- Submit jobs (JOB)
- Set job parameters (JOB, $$ JOB)
- Modify, cancel, abort, or restart jobs (JOB)

- Monitor subsystem usage (BATCH)
- Monitor queue characteristics and availability (BATGEN)

These operations are described below.

SUBMITTING BATCH JOBS

To submit a job, use the command:

```
                 ┌ -ACCT information                 ┐
                 │ -ARGS cpl-arguments               │
                 │ -CPL                              │
                 │          ( seconds )              │
                 │ -CPTIME {           }             │
                 │          ( NONE    )              │
                 │          ( minutes )              │
JOB pathname-1   │ -ETIME  {           }             │
                 │          ( NONE    )              │
                 │ -FUNIT number                     │
                 │ -HOME pathname-2                  │
                 │ -PRIORITY value                   │
                 │ -QUEUE queue-name                 │
                 │           ( YES )                 │
                 │ -RESTART {       }                │
                 └           ( NO  )                 ┘
```

Batch will then send a "job submitted" response announcing the job's job-id number and reminding the user (if he didn't use the -HOME option) of the home UFD for the job. For example:

```
OK, job pnjob
[JOB rev 18.1]
Your job, #00015, was submitted to queue Normal-1.
Home=<FOREST>BEECH>BRANCH4>TWIG
```

As this example shows, jobs may be submitted without options. The Batch monitor places these jobs in the first available queue and uses that queue's default values for all necessary parameters. On the other hand, users may specify queue and/or parameters, using the JOB command's options as described below.

Note

All numbers must be decimal integers.

Option	Description
-ACCT information	Allows the user to specify accounting information for his job. The information must be 80 characters or less in length. It may not be an explicit register setting (octal number) or be preceded by an unquoted minus sign. If the information contains spaces, commas, or comment delimiters (/*) it should be enclosed in apostrophes. (For example: -ACCT 'OK, HERE WE GO'). The information will be included in job DISPLAYs, but will not be used in running the job.
-ARGS cpl-args	Used to pass CPL arguments to the job being processed. -ARGS must be the last option issued on a command line as everything that follows the -ARGS option on the command line (except comments) is assumed to be the CPL arguments being passed. JOB doesn't read the CPL arguments; it just passes them to the CPL file when execution of the file begins.
-CPL	Runs submitted file as a CPL file, no matter what the file's name is.
-CPTIME $\left\{\begin{array}{l}\text{seconds}\\ \text{NONE}\end{array}\right\}$	Specifies the maximum amount of CPU time (in seconds) to be allotted to the job. NONE requests that no time limit be placed on the job. If the job exceeds the time limit, it will be aborted.
-ETIME $\left\{\begin{array}{l}\text{minutes}\\ \text{NONE}\end{array}\right\}$	Specifies (in minutes) the elapsed time to be allowed before the job is aborted. Details are the same as those for -CPTIME.
-FUNIT number	Specifies the file unit to be used for command input. Permissible values range from 1 to 16, to 1 to 126, depending on the limit set by the System Administrator. Default depends on the queue to which the job is submitted. It is usually 6. -FUNIT may not be used in CPL jobs, as CPL jobs receive dynamically assigned file units. Attempts to use -FUNIT result in the following message: `Illegal combination. -FUNIT (JOB)` A similar message is received, but either the -ARGS or the -CPL option appears on the same command line as -FUNIT.

-HOME pathname	Specifies the UFD in which a 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 (i.e., it may not begin with *>), and may not exceed 8Ø characters in length.
-PRIORITY value	Determines the job's priority within its queue. Possible values are from Ø to 9, with 9 being the highest (most favored) priority. The default depends on the queue.
-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.)
-RESTART $\begin{Bmatrix}\text{YES}\\\text{NO}\end{Bmatrix}$	Determines whether a job can be restarted following an ABORT or a system shutdown. The default is always YES.

If, for any reason, the Batch monitor cannot accept the job as submitted, it will send the user error messages containing the information he needs to resubmit the job successfully. These messages are listed in Appendix A; they are generally self-explanatory.

SUPPLYING OPTIONS VIA THE $$ COMMAND

Any or all of the JOB command's eight options may be given in the first non-comment line of the command file itself by the command:

$$\text{\$\$ JOB} \quad \begin{Bmatrix}*\\\text{username}\end{Bmatrix} \quad \{\text{options}\}$$

If a specific <u>username</u> is given on the $$ command line, only a user logged in with that username can submit the file. If an asterisk (*) is used instead, any user can submit the file.

Users will probably find the $$ command handiest for parameters they expect to remain constant whenever the job is submitted, and the JOB command options handiest for parameters which change from submission to submission.

Parameters given in the $$ JOB command line may be overridden by giving a different value for the same parameter in the JOB command. For instance, if you specified "$$ JOB RESEARCH -CPTIME NONE" in your file, but wanted to run the job in a queue which had a CPU time limit, the command "JOB TEST_SCORES -CPTIME 18Ø -QUEUE FAST" would run the job in queue FAST with a CPU time limit of 18Ø seconds.

Note

With one exception, any Batch command file, even one including a $$ JOB command, can be run interactively.. The exception is a file using the $$ JOB -HOME option. When run interactively, the $$ JOB line will be ignored, and no ATTACH will be done. In this case, add an ATTACH command to the file immediately following the $$ JOB line.

CONTROLLING BATCH JOBS

Modifying Parameters

To modify a job's parameters after it has been submitted, use the -CHANGE option of the JOB command:

```
                                 ⎧ -ACCT      information ⎫
    ⎧jobname⎫                    ⎪ -CPTIME    ⎧seconds⎫   ⎪
JOB ⎩job-id ⎭ -CHANGE            ⎪            ⎩NONE   ⎭   ⎪
                                 ⎪ -ETIME     ⎧minutes⎫   ⎪
                                 ⎨            ⎩NONE   ⎭   ⎬
                                 ⎪ -FUNIT     number      ⎪
                                 ⎪ -HOME      pathname    ⎪
                                 ⎪ -RESTART   ⎧YES⎫       ⎪
                                 ⎩            ⎩NO ⎭       ⎭
```

For example:

```
JOB #10039 -CHANGE -ACCT 'research' -HOME ECON>STATS
JOB TEST_SCORES -CHANGE -FUNIT 8 -RESTART YES
```

A job's -QUEUE and -PRIORITY options cannot be CHANGEd. If they are in error, the job must be CANCELed and resubmitted.

Restarting Jobs

Users wishing to CHANGE jobs which are already running can do so by following a JOB -CHANGE command with a JOB -RESTART command. For example:

```
JOB TEST_SCORES -CHANGE -HOME RESRCH>STATS>NEWSTATS
JOB TEST_SCORES -RESTART
```

This procedure will mark the changes in the job's status, terminate execution, and then flag the job as ready for restarting under its new conditions.

Note

Distinguish between the -RESTART YES/NO option and the -RESTART command. The option always takes an argument; it signals whether or not a job may be restarted. The -RESTART command takes no argument; it attempts to abort and restart the job.

Cancelling Jobs

To prevent a waiting or held job from running, use the command:

```
    {jobname}
JOB {job-id } -CANCEL
```

This command will not halt a job that is already running; but it will mark that job as unrestartable.

Aborting Jobs

To terminate execution of a job already running, use:

```
    {jobname}
JOB {job-id } -ABORT
```

This command cancels a waiting or held job and forces a running job to log itself out immediately.

The JOB -CHANGE, -CANCEL, -ABORT, and -RESTART commands will accept a filename in place of a job-id only if that filename is unique among the user's active jobs. Thus, if file TEST has been submitted once, the command "JOB TEST -CANCEL" will work. But if two submissions of TEST (for example, #1ØØ57 and #1ØØ64) are active, you must use the job-id to tell the monitor which job to cancel. The monitor accepts only one command at a time; JOB TEST -ABORT -RESTART is illegal, as is JOB #1ØØ35, #1ØØ39 -CANCEL.

MONITORING BATCH

Users may monitor their own jobs within the Batch system by using the JOB -STATUS and JOB -DISPLAY commands; they may monitor subsystem usage through the BATCH -DISPLAY command; and they may monitor the characterisitics and availability of queues through the BATGEN -DISPLAY and BATGEN -STATUS commands. These commands work as follows:

```
▶ JOB [job-id  ]  {-STATUS }
      [jobname ]  {-DISPLAY}
```

Monitors the progress of the user's own jobs. The -STATUS and DISPLAY options govern the amount of information to be shown, while the jobname and job-id options allow the user to specify the jobs on which he wants information, as follows:

Option	Description
job-id	A 5-digit number assigned to a job by the monitor when the job is placed in a queue. Use the job-id to request information on one job only.
jobname	The name of the file being run. If the job was submitted as a pathname (e.g., JOB FELLOWSHIP>HOBBITS>FRODO), its jobname is the final element of the pathname (e.g., FRODO). Use this format to request information on multiple submissions of a file.

(Omitting jobname and job-id requests information on all the user's active jobs.)

-STATUS	Prints out the job's jobname and job-id, the name of the queue in which it is placed, and its execution status: whether it is held, waiting, running, completed, or aborted.
-DISPLAY	Provides status information and values for all JOB and $$ JOB command options (except for "-HOME") -- both those specified by the user and those assumed from queue-defined defaults.

Using the MESSAGE Command

Another way to monitor your Batch jobs is to have the jobs send messages to your terminal announcing the completion of key portions of the job. To do this, use the MESSAGE command (explained in Section 14), as shown below.

Messages from CPL Programs: CPL programs put their messages in &DATA groups. The format is:

```
                   {user-number}
&DATA MESSAGE      {UFD-name   }
   text of message
&END
```

For example:

```
    &DATA MESSAGE BEECH
       Customer list update completed
    &END
```

<u>Messages from COMINPUT Files</u>: Command input files write the text of their messages as comment lines:

```
                {user-number}
    MESSAGE     {UFD-name   }
    /* text of message
```

For example:

```
    MESSAGE BEECH
    /* Customer update completed
```

Using this format prevents errors from occurring if the recipient of the message is not logged in at the time the message is to be sent.

▶ BATCH -DISPLAY

Monitors Subsystem Usage. It prints the number of jobs waiting in each queue and lists all jobs currently executing, identifying them by user, job-id, phantom user-number, and queue. For example:

```
    OK, batch -display
    [BATCH rev 18.1]

    Number of waiting and held jobs:

    Queue     Jobs
    -------- ----
    Normal-2 76

    Currently running jobs:

    User   Jobid# #  Queue
    ------ ------ -- ------
    TURNER #10032 60 Normal-2
    BURLEY #00172 62 Normal-1
```

▶ BATGEN -STATUS

Lists the currently defined queues and notes whether each is blocked (not accepting jobs) or unblocked (available for use).

▶ BATGEN -DISPLAY [queuename]

Identifies and gives full characteristics for each queue, if queuename is not specified. If queuename is specified, gives characteristics for that queue only. For example:

```
OK, batgen -display normal
[BATGEN rev 17.2]

Queue name = normal, unblocked.
Default cptime=30, etime=None, priority=5;
Maximum cptime=180, etime=None; Funit=6;
Delta rlevel=1; Timeslice=20;
```

In this example, normal is the queue's name. Unblocked means that the queue is accepting jobs for queueing and execution. The default cptime and etime values will apply to jobs that don't specify their own CPU time or elapsed time options. The maximum cptime and etime values are the largest allowed for any job running from the queue. Priority and funit are default values for those options.

Delta rlevel and timeslice refer to run-time 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 queue in the example is designed for average jobs.

Note

If the System Administrator has not read-enabled the BATDEF file, the BATGEN commands will return error messages. In this case, users needing information about queues should see their supervisor, the operator, or the System Administrator.

Part III
System Facilities

SECTION 11

FILE-HANDLING UTILITIES

INTRODUCTION

This section introduces you to Prime's basic file handling utilities. These utilities allow you to:

- Sort one or more unsorted files into one sorted file (SORT)
- Merge several sorted files into one sorted file (SORT)
- Compare files with each other (CMPF)
- Resolve differences between files (MRGF)
- Move files and subdirectories between directories (FUTIL)
- Copy or delete entire directories (FUTIL)

SORTING FILES (SORT)

The SORT command sorts up to 20 files, on up to 50 keys, into a single output file. SORT preserves the order of input for records with equal keys (i.e., it is a stable sort).

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. _The PRIMOS Commands Reference Guide_ contains complete information and sorting instructions for each file type.

SORT can also sort files using the EBCDIC collating sequence. For details, see _The PRIMOS Commands Reference Guide_.

Using SORT

To use SORT, provide information in a three- or four-step sequence, as follows:

1. Give the SORT command.
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 -MERGE is specified, enter additional filenames.

SORT normally specifies the information it wants at steps 2, 3, and 4. However, once you are familiar with the prompt dialog, you can suppress the printout by using the -BRIEF option with the command line. If -BRIEF is specified, simply give the information line by line in the same order SORT asks for it. Refer to the sample sort that concludes this discussion 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:

```
SORT [-BRIEF] [-SPACE] [-MERGE] [-TAG   ]
                                [-NONTAG]
```

SORT's options are as follows:

Option	Meaning
-BRIEF	SORT program messages are not printed at the users terminal.
-SPACE	Any blank lines are deleted from the SORT output file.
-MERGE	A merge of presorted files is requested.
-TAG	A TAG sort (described below) is requested.
-NONTAG	A NONTAG sort (described below) is requested.

A TAG sort is specified when large files are sorted. For unordered files it is a faster sort than NONTAG. Internally, the TAG sort stores input records separate from the key data. After all keys have been sorted and merged, the corresponding records are then located and output.

A NONTAG sort may be specified 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.

If neither -TAG nor -NONTAG is specified, the system defaults to TAG.

Note

Output files may be a different type than input files.

SORT responds by requesting:

- 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. To specify this sort, simply 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. If the data within a key field are to be sorted by some code other than straight ASCII, type a space and the data type after the ending column. (The SORT dialog will list data types and their codes. They are also explained, in greater detail, in The PRIMOS Commands Reference Guide.) If the sort on any key is to be done in reverse (descending) order, type a space and an "R" after the ending column or data type. For example, to sort a list of names and addresses, the entire entry of 80 characters might constitute the sort field, and the commands would run:

```
OK, SORT -BR
JUMBLED.NAMES   NEAT.NAMES
1  80
```

Unless the -MERGE option has been specified, sorting begins when the last pair of column numbers is entered. When the sort is complete, SORT prints at the terminal the number of passes needed for the sort and the number of items (i.e., lines) placed in the output file, and then returns to PRIMOS.

Other File Specifications

If you are sorting more than one file, give all filenames plus the number of keys 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-REV18.1]

    OK,
```

If you are sorting uncompressed or fixed length files, or if you are sorting binary files using ASCII keys, you will have to specify additional file information (via keywords) along with the filenames. See The PRIMOS Commands Reference Guide for details.

Key Specifications

SORT recognizes 13 types of keys. ASCII files (compressed and uncompressed) can use seven of them: A and AU for alphanumeric data, U, LS, TS, LE and TE for numeric data.

Alphanumeric keys: The two alphanumeric keys are ASCII (A), which sorts in a strict ASCII sequence, and ASCII, upper and lower (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 keys for ASCII sorts are:

- U Numbers without plus or minus signs
- LS Numbers preceded by plus or minus signs (Numbers without signs are considered positive.)
- TS Numbers followed by plus or minus signs (Numbers without signs are considered positive.)

(The LE and 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 -br
numbers numbers.1
1 10 ls

BEGINNING SORT



 PASSES      2          ITEMS       7

[SORT-REV18.1]

OK, slist numbers.1
-9999
-8205
-6783
 4114
+5483
 8265
+9765

OK,
```

Additional Filenames for MERGE Operation

After key fields have been specified using 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 merge is complete, SORT prints the number of passes and returns to PRIMOS.

A Mergesort Example

Here is an example of a mergesort. Assume we have created two transaction files, in which each line (record) has the following format: a transaction number in columns 1-5, a credit or debit notation in column 6, a customer name in columns 8-17, a customer ID number in columns 19-25, 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). Now we are going to merge the two files, sorting on the same three keys. The sort, with the full SORT dialog, 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-REV18.1]

    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,
```

FILE COMPARISON (CMPF)

The PRIMOS command CMPF permits the simultaneous comparison of up to five ASCII files of varying lengths. The format is:

```
    CMPF file-1 file-2 [.....file-5] [options]
```

The first file, file-1, is treated as the original file against which the other files are compared. The CMPF command produces output indicating which lines have been added, changed, or deleted in the other files.

The options which may be specified are:

Option	Function
-BRIEF	Suppresses the printing of differing lines of text of files being compared. Only identification letters and line numbers are printed.
-MINL number	Sets the minimum number of lines that must match after a discrepancy between files is found. Needed in order to resynchronize file comparison. Default = 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).

After a difference between the original file and another specified file has been discovered, 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 in 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 line number of the line containing discrepancies. The corresponding lines of other files are indicated in the same manner using letters B through E respectively.

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
GO

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 (MRGF)

The MRGF command merges up to five ASCII files. The format is:

```
MRGF file-1 [file-2 ...file-5] -OUTF outfile [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, outfile. When corresponding lines of text in the files differ, the user is asked by the MRGF program to solve the conflicts. This is done by entering an interactive mode in which the user can specify the contents of the output file. In this mode, the command x (x = A-E) causes all the queried lines from file X to be inserted; the command xn causes line n from file X to be inserted. New text can be inserted by entering a blank line at the terminal (thus sending MRGF into 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 will be done. The lines must be entered character-for-character as they are to appear.

The options taken by the MRGF command are similar to those for the CMPF command. There is an additional option, -FORCE, which causes file-2 to be the preferred file if conflicts exist between several files. No MRGF interactive dialog will be generated when conflicts arise if the -FORCE option is used. File-2 is assumed "correct" and the other files forced to comply with it.

FILE UTILITY (FUTIL)

FUTIL is a file utility command for copying, deleting, and listing files and directories. FUTIL is most often used for copying files and directories from one directory to another. It is also useful for deleting groups of files and entire directories. Its list option allows the user to examine file and directory properties and to keep track of the contents of directories involved in the copy or delete processes. FUTIL allows operations on files within User File Directories (UFDs) and segment directories.

Invoking FUTIL

To invoke FUTIL, type FUTIL. When ready, FUTIL prints the prompt character, >, and waits for a command string from the user terminal. FUTIL accepts either upper- or lowercase input, but passwords must be entered exactly as they have been created. (Most other commands will convert passwords to uppercase before attempting the match. FUTIL does not.) To abort long operations (such as LISTF), type BREAK, and restart FUTIL by typing S 1000.

To use FUTIL, type one of the FUTIL subcommands (listed below) followed by a carriage return, and wait for the prompt character before issuing the next command. The erase (") and kill (?) characters are supported in both command and subcommand lines

FUTIL Commands

Below are some examples of the most commonly used FUTIL commands. An overview of FUTIL commands appears at the end of this section. For complete details on all the FUTIL commands, which are summarized at the end of this section, see The PRIMOS Commands Reference Guide.

Copying Files and Directories

FUTIL provides several commands which allow the user to copy files, directories, or directory trees. These commands, their functions and formats are listed below:

Command	Function
COPY	Copies files (as many as will fit on line).
TRECPY	Copies directory trees.
UFDCPY	Copies entire UFD structure (complete with all files).
TO	Specifies directory to which file(s) or directories are to be copied. Accepts a pathname. Default is home directory.
FROM	Specifies directory from which files or directories are to be copied. Accepts a pathname. Default is home directory.

The general formats of these comands are:

```
COPY filename [new-name],[filename new-name]

TRECPY filename

UFDCPY
```

Copying Files: In order to copy a file, the user must have read access rights. The name of a file may be changed by indicating the desired new name immediately after the current name has been specified. Filename pairs are separated by commas on the command line.

Situation 1: Suppose we want to copy the files HITS and MISSES from the directory NAUTILUS into our current directory, SECRETS. The pathname of SECRETS is represented as follows: <*>SECRETS.

In pathnames, <*> represents the current disk. In this case, it represents disk 2. This pathname can also be represented as <MONITOR>SECRETS. MONITOR is the volume-name of the logical device, whereas 2 is the volume-number. The volume-name and number can be used interchangeably in a pathname, and both appear in the following examples. Any directory subordinate to SECRETS would be described by a relative pathname, as in, *>DOMESTIC. In relative pathnames, the use of * indicates the current directory.

To move files from any directory to the current directory, the following general steps are taken:

1. Invoke FUTIL.
2. Define the FROM directory.
3. Define the files to be copied and indicate new filenames (optional).

The FUTIL dialog for this particular situation is:

```
OK, FUTIL
[FUTIL rev 18.1]
>FROM <1>MARINE>NAUTILUS
>COPY HITS, MISSES ZEROES
>QUIT
OK,
```

The files HITS and ZEROES (formerly MISSES) are now in our current directory SECRETS, as well as in the FROM directory NAUTILUS. Notice that a TO directory was not specified. If the TO directory is not explicitly indicated, FUTIL assumes it to be the current directory. Although the file MISSES is called ZEROES in the current directory, its name is not changed in the original (or FROM) directory.

Situation 2: Suppose we want to copy all the contents of the directory HOLLAND to another directory CLASSIFIED, on the current disk. The files and directories contained in HOLLAND are called a directory tree. The FUTIL dialog would be as follows:

```
OK, FUTIL
[FUTIL rev 18.1]
>FROM <1>MARINE
>TO <*>CLASSIFIED
>TRECPY HOLLAND
>Q
```

This copies the directory HOLLAND (with its subordinate files and directories) to the directory CLASSIFIED. The <*> indicates the current disk. HOLLAND is now a subdirectory in CLASSIFIED.

Situation 3: Suppose we wish to copy the entire directory tree MARINE into the UFD REPORTS. The FUTIL dialog would be:

```
OK, FUTIL
[FUTIL rev 18.1]
>FROM <NAVY>MARINE
>TO <MONITOR>REPORTS
>UFDCPY
>Q
```

The entire batch of files and directories listed under the UFD MARINE are now listed as a subdirectory under the UFD REPORTS.

Situation 4: We can also copy files from our home (current) directory to another. It is not necessary to specify a FROM name. In the absence of a FROM specification, FUTIL assumes the FROM directory to be the current working directory. Simply specify the directory to which the files are to be copied.

The current directory in this situation is NAUTILUS. FUTIL allows you to move to other directories with the ATTACH subcommand, abbreviated "A". It is not necessary to return to PRIMOS in order to change the working directory location. For example:

```
OK, FUTIL
[FUTIL rev 18.1]
>A <1>MARINE>NAUTILUS
```

The directory NAUTILUS is now the current working directory. To copy the file HITS from the current directory up to the directory MARINE, do the following:

```
>T <1>MARINE
>C HITS
>Q
```

Deleting Files and Directories

Commands for deleting files, directory trees and UFDs are:

Command	Function
DELETE	Deletes specified files from FROM directory.
TREDEL	Deletes specified directory trees or segment directories, including MIDAS files, from FROM directory.
UFDDEL	Deletes entire specified UFD.

The user must have read, write, delete/truncate access rights to delete any file. Below are some situations in which FUTIL is used to delete several types of files and directories.

Situation 1: In order to delete the file HITS from the sub-UFD NAUTILUS, the following dialog could be used:

```
OK,FUTIL
[FUTIL rev 18.1]
>FROM <NAVY>MARINE>NAUTILUS
>DELETE HITS
>Q
```

Situation 2: If we wanted to delete the directory tree rooted in the sub-UFD HOLLAND, we would do the following:

```
OK,FUTIL
[FUTIL rev 18.1]
>FROM <1>MARINE
>TREDEL HOLLAND
>Q
OK,
```

This deletes the directory HOLLAND and its entry in MARINE. Similarly, to delete segment directories and MIDAS files, use the TREDEL option, as shown.

Situation 3: To delete the contents of CLASSIFIED appearing on the current disk, (2), the following dialog could be implemented:

```
OK,FUTIL
[FUTIL rev 18.1]
>FROM <*>CLASSIFIED
>UFDDEL
>QUIT
OK,
```

This deletes all subordinate directories and files from the UFD CLASSIFIED. The directory itself, however, is not deleted.

Listing Contents of a Directory

The LISTF command in FUTIL displays a list of all the files and directories in the FROM directory. It also displays the FROM directory pathname and the TO directory pathname (default). The various options to the LISTF command provide information on all the files contained in the FROM directory.

FUTIL COMMAND SUMMARY

A̲TTACH pathname

Changes working directory to pathname.

CLEAN prefix [level]

Deletes files beginning with prefix, for indicated number of levels (default=1).

C̲OPY from-name [to-name] [,from-name [to-name]] ...

Copies named files from FROM directory to TO directory. If to-names are omitted, copies have same names as originals.

C̲OPY (from-position) [(to-position)]

Copies from one segment directory to another. If to-position is omitted, copy goes to same position as original. Note that COPY from-name (to-position) and COPY (from-position) to-name are also legal.

COPYDAM

Same as COPY but sets file type of copy to DAM.

COPYSAM

Same as COPY but sets file type of copy to SAM.

```
CREATE directory [owner-password [non-owner-password]]
```

Creates directory in current TO directory (with optional passwords).

```
DELETE   file-a [file-b] ...
         (position-a) [(position-b)] ...
```

Deletes from FROM directory, named files or, in segment directories, deletes files at specified positions.

```
FORCE { ON    }
      { [OFF] }
```

ON forces read-access rights in FROM directory for LISTF, LISTSAVE, SCAN, UFDCPY, and TRECPY. OFF stops FORCE action (default).

```
FROM pathname
```

Defines FROM directory for subsequent commands such as COPY, LISTF, etc.

```
LISTF [level] [FIRST] [SIZE] [PROTEC] [RWLOCK] [TYPE]
        [DATE] [PASSWD] [LSTFIL]
```

Lists files and attributes at terminal (and into optional file called LSTFIL).

```
LISTSAVE filename [level] [FIRST] [SIZE] [PROTEC] [RWLOCK]
        [TYPE] [DATE] [PASSWD]
```

Same as LISTF, with the LSTFIL option specified, but writes output to filename.

```
PROTEC filename [owner-access [non-owner-access]]
```

Sets protection attributes for filename.

```
SCAN filename [level] [FIRST] [LSTFIL] [SIZE] [PROTEC]
        [RWLOCK] [TYPE] [DATE] [PASSWD]
```

Searches FROM directory tree for all occurrences of specified filename and prints requested attributes.

SRWLOC filename lock-number

    Sets per-file read/write lock.

TO pathname

    Defines TO directory for subsequent commands such as CREATE and all copying commands.

TRECPY directory-a [directory-b] [,directory-c [directory-d]] ...

    Copies directory tree(s) in FROM directory into TO directory.

TREDEL directory-a [directory-b] ...

    Deletes directory tree(s) in FROM directory.

TREPRO pathname [owner-access [non-owner-access]]

    Sets protection rights for directory and contents (default 1 Ø).

TRESRW pathname lock-number

    Sets per-file read/write lock for all files in pathname.

UFDCPY

    Copies entire FROM directory into TO directory.

UFDDEL

    Deletes entire FROM directory.

UFDPRO [owner-access [non-owner-access [level]]]

    Sets protection attributes for entire FROM directory.

UFDSRW lock-number [n-levels]

Sets per-file read/write lock for n-levels in FROM directory.

Lock-number	Meaning	Code
Ø	Use system read/write lock	SYS
1	n readers OR 1 writer	W/NR
2	n readers AND 1 writer	1WNR
3	n readers AND n writers	NWNR

SECTION 12

USING TAPES AND CARDS

ACCESSING DATA ON TAPES AND CARDS

Existing source programs resident on punched cards, magnetic tape, or punched paper tape can easily be read into disk files using PRIMOS-level utilities. In addition, the punched card and magnetic tape transfer utilities will translate from BCD or EBCDIC representation into ASCII representation saving considerable time and effort.

Subroutines and other installation-dependent operations may be altered to conform to PRIMOS by using the Editor (ED) described in Section 4.

The general order of operations for input from a peripheral device is:

1. Obtain exclusive use of the device (Assigning).
2. Transfer programs with appropriate utility.
3. Relinquish exclusive use of the device (Unassigning).

Assigning a Device

Assigning a device gives the user exclusive control over that peripheral device. The PRIMOS-level ASSIGN command is given from the terminal:

```
ASSIGN device [-WAIT]
```

device is a mnemonic for the appropriate peripheral:

CRn	Card Reader n (n=0,1)
MTpdn [-ALIAS MTldn]	Magnetic Tape Unit pdn (pdn=0-7)
MTX -ALIAS MTldn	Any Magnetic Tape Unit (ldn=0-7)
PTR	Paper Tape Reader

-WAIT is an optional parameter. If included, it queues the ASSIGN command if the device is already in use. The assignment request remains in the queue until the device becomes available or the user types the BREAK key at the terminal; both occurrences return the user to PRIMOS. If the requested device is not available and the -WAIT parameter has not been included, the error message:

```
The device is in use. (ASSIGN)
```

will be printed at the terminal.

After all I/O operations are completed, exclusive use is relinquished by the command:

```
    UNASSIGN device
```

device is the same mnemonic used in the ASSIGN command.

READING PUNCHED CARDS

Assign use of the parallel interface card reader by:

```
    AS CRn -WAIT
```

To read cards from the card reader, load the card deck into the device and enter the command:

```
    CRMPC deck-image   [-PRINT] [-CRØ] [-CR1]
```

deck-image	The pathname of the file into which the card images are to be loaded.
-PRINT	Print card while reading.
-CRØ	Use device CRØ (default).
-CR1	Use device CR1.

Source deck header control cards are set up as follows:

Source deck representation	Columns 1 and 2 of deck header card
BCD	$6
EBCDIC	$9
ASCII	no header card

Reading continues until a card with $E in columns 1 and 2 are encountered (end of deck); control returns to PRIMOS and the file is closed. If the cards are exhausted (or the reader is halted by the user), control returns to PRIMOS but the file is not closed. If more cards are to be read into the file, the reader should be reloaded; reading is resumed by the command START at the terminal.

Close the file with the command:

```
    CLOSE ALL

      or

    CLOSE deck-image
```

Example of card reading session:

```
    OK, AS CR -WAIT
    OK, CRMPC old-program-1
    OK, UN CRØ
    OK,
```

If a serial interface card reader is used, the process is similar, with slightly different reader commands:

```
    OK, AS CARDR -WAIT
    OK, CRSER old-program-2
    OK, UN CARDR
    OK,
```

CARDR may be abbreviated to CAR.

READING PUNCHED PAPER TAPE

First load tape into reader; then assign tape reader. Source programs punched on paper tape in ASCII representation can be read into a disk file with the Editor.

```
    OK, AS PTR -WAIT    Assign tape reader
    OK, ED              Invoke Editor
    INPUT
     (CR)               Switch to EDIT mode
    EDIT
     INPUT (PTR)        Input from tape reader
    EDIT                Tape is being read
    FILE filename       File input under filename
    OK, UN PTR          Unassign tape reader
```

MAGNETIC TAPE UTILITIES

The Prime magnetic tape utilities (MAGNET, MAGRST, and MAGSAV) allow the duplication of magnetic tapes, the transfer of files from disk to tape and vice-versa, and the transfer and translation of tapes in selected non-Prime formats to and from PRIMOS disk files. All mag tape operations done with these utilities require the assignment of at least one magnetic tape drive unit.

Assigning Tape Drives

Magnetic tape drive assignment can be set up at each installation by the System Administrator in one of three ways:

- 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 terminal indicating the status of the assignment request.

- Tape drive assignment from any user terminal is strictly forbidden. This feature is used to restrict access to tape drives in security-conscious environments, or when the operator is not available to process requests.

The ASSIGN Command Format

Users may assign magnetic tape drives in any one of three ways:

- By physical device number (pdn):

```
ASSIGN MTpdn [-options]
```

- By logical device number (ldn):

```
ASSIGN MTX -ALIAS MTldn
```

- By logical device number plus characteristics:

```
ASSIGN MTX -ALIAS MTldn -options
```

Assigning a drive by physical device number requests that particular drive. If the drive is busy, -WAIT queues the request. Assigning a drive by logical device number says, "Give me any tape drive, and call it number ldn." (The -ALIAS option supplies the number.) Any free tape drive may then be assigned. If all devices are busy, -WAIT queues a request for the first free device. Assigning a drive by logical number plus characteristics asks for any drive that can handle a particular type of tape (for example, a 9-track tape at 6250 bpi), and gives the drive a logical alias. In all three cases, users will be told which physical device has been assigned to them; they may refer to the device by either its physical number or its logical alias. Additionally, ASSIGN allows special requests to be made of the system operator; for example, removing the WRITE-ring or mounting a particular tape. (This version of the ASSIGN command applies only to mag tape drives;

other peripheral devices like the paper tape reader (PTR) cannot be assigned with the options described here.) The command format, complete with optional arguments, is:

```
ASSIGN {MTpdn [-ALIAS MTldn]} [-option(s)]
       {MTX    -ALIAS MTldn }
```

The arguments and options are:

Argument	Description
MTpdn	Mag tape (MT) unit number from 0 to 7, inclusive. pdn is the physical device number assigned to each drive at system startup. Numbers can be obtained from the system operator.
MTX	Tells the operator to assign "any available drive"; MUST be accompanied by -ALIAS MTldn, which assigns a number (alias) to the drive for reference purposes. See below. The actual drive assigned depends on any other options which appear on the command line.
-ALIAS MTldn	The logical drive number, from 0 to 7, inclusive. ldn is a user-specified number assigned to a particular physical drive unit; used as an alias for the pdn in subsequent mag tape operations. Logical and physical device numbers can be used interchangeably in MAGNET, MAGSAV and MAGRST dialogs; however, to avoid confusion, give MAGRST/MAGSAV the logical device number, if you're using aliases. See Note below.
-WAIT	Indicates user is willing to wait until requested drive is available.
-TPID id	Requests the operator to mount a particular reel of tape, identified by a tape id; requires operator intervention. id is a list of tape identifiers (arguments) describing a particular reel of tape, and/or type of tape drive (name, number, etc.). Identifiers may not begin with a hyphen (-) which is a reserved character indicating the next control argument on the ASSIGN statement line.

-RINGON -RINGOFF	Protection rights may be specified by:

RINGON or	Read and write permitted.
RINGOFF	Read only; write-protection in effect.

	Requires operator intervention for removal or replacement of write-ring.
-800BPI -1600BPI -6250BPI	Particular tape density settings are requested with these options. Most drives can handle 800 and 1600 bpi settings. Requires operator intervention.
-7TRK -9TRK	Indicates 7- or 9-track tape drive; default is 9-track. Requires operati intervention if -7TRK is specified.

Using the -ALIAS Option

The -ALIAS option is useful in several general situations:

- When you request special features and do not know which available drive meets the stated requirements
- When you are writing a command file to perform mag tape operations and have no way of knowing which tape drive is available at a given time
- When you know the actual pdn of the drive being assigned but prefer to give it another number, for ease of reference, or to avoid confusion

Once an alias has been assigned, either the physical or logical device number can be used to refer to the drive in question in subsequent mag tape drive operations like MAGSAV. The logical device number is "mapped into", or associated with the physical device number in an internal table.

With the MTX option, command files which perform mag tape operations can be executed independently of a particular drive's availability. The arbitrary number assigned the tape drive with MTX -ALIAS can be used in writing responses to the dialog of the utility invoked by the command file.

<u>Note</u>

MAGSAV and MAGRST ask the user for the device number of the drive on which a tape is mounted. Both dialogs assume the number given is a logical device number: consequently, the internal list of logical device numbers is searched first. If a match is found, MAGSAV/MAGRST will interact with the tape mounted on the corresponding physical drive. Suppose the user first assigns physical device MTØ as logical MT1, then assigns physical MT1 as logical MTØ. If the user answers "1" to the "TAPE UNIT:" prompt of MAGSAV (or MAGRST), the utility assumes that "1" is a logical device number (ldn). Thus, it attempts to read from or write to, as the case may be, the tape mounted on physical device MTØ, which the user previously assigned as logical MT1.

USING ASSIGN

The following examples illustrate some uses of ASSIGN. In all cases, the distinction between what the user can do <u>without</u> operator intervention and what must be done <u>with</u> operator assistance is indicated.

<u>Default Assignment</u>

The standard form of assignment does not require operator intervention on systems with the default configuration (user-privileges allowed). For example:

```
OK, AS MT1
Device MT1 Assigned.
```

Mag tape drive MT1 is assigned. (1 is the physical device number.) If the device is currently assigned to another user or process, this message appears:

```
The device is in use.  (ASSIGN)
ER!
```

On systems where all mag tape requests are monitored, the request above would be acknowledged with the same message, but a slight delay would be observed. The operator has to answer each request, which results in a delayed response at the user terminal.

Logical Aliases

Logical device numbers can be assigned by the user without operator assistance on default privilege systems, providing that no other special requests are made on the same ASSIGN command line:

```
OK, AS MT1 -ALIAS MT0
Device MT1 Assigned.
```

Note that the physical, not the logical, device number is returned.

Physical device MT1 can now be referred to as logical device MT0. ldn's and pdn's are associated internally in a special table and can be used interchangeably. If no ldn alias is requested, the default logical device number is the same as the physical device number of the drive. The STAT DEV command lists the physical-to-logical number correspondence:

```
OK, STAT DEV

DEVICE  USRNAM  USRNUM  LDEVICE
MT1     DOUROS    7     MT0
```

If no logical alias had been requested, the LDEVICE entry would be identical to the DEVICE entry; in this case, MT1.

Aliases in Operator Mode

Similarly, logical aliases can be requested on operator-controlled systems. Again, the pdn of the assigned device will be displayed at the user's terminal with a message of this general form:

```
Device MTpdn Assigned.
```

pdn varies with the actual physical device chosen by the operator.

Special Requests

If control arguments for special requests appear on the ASSIGN command line, then the operator must intervene, even on systems with default user privileges. For example, all ASSIGN commands with the MTX option must be handled by the operator:

```
ASSIGN MTX -ALIAS MT4
```

The operator is requested to assign any available tape drive as logical device 4. A message is displayed at the user's terminal, indicating which physical drive has been assigned.

The operator must also intervene if a user wants a tape mounted, or if a particular density setting is required, or if a particular drive is needed (for instance, to read a tape recorded at 6250 bpi). For example:

```
    AS MTX  -ALIAS MT3  -TPID POWER  -9TRK  -RINGOFF  -6250
```

The operator is requested to mount the "POWER" tape on a 9-track drive that can handle 6250 bpi. In this case, "POWER" is the name written on the tape reel to identify the tape and is not necessarily the recorded label. In addition, the user wants write-protection and is assigning an alias of MT3 (ldn) to whatever device the operator chooses. This request, if processed, might be acknowledged with this display:

```
    Device MT0 Assigned.
```

Operator Not Available

If the operator is not available to handle requests, any attempt by a user to assign a mag tape drive will result in this message:

```
    OK, AS MT1
      No MagTape Assignment Permitted. (AS)
    ER!
```

Operator Can't Handle

If any request cannot be handled by the operator for any reason, the following message appears at the terminal:

```
    OK, AS MTX -ALIAS MT0 -6250
      MagTape Assignment Request Aborted (ASSIGN)
    ER!
```

Improper Use of ASSIGN

Should an improper form of the ASSIGN command be issued, an error message appears, as well as the proper command format, complete with all the options. For example:

```
OK, AS MT1 -ALIS MT0 -RINGOFF
"-ALIS" not implemented or improper use of argument. (ASSIGN)
Usage: ASSIGN MTn [-ALIAS MTm] [<options>]
       ASSIGN MTX  -ALIAS MTn  [<options>]
       Options: [ -TPID <id> ] [ -7TRK | -9TRK ] [ -RINGON | -RINGOFF ]
                [ -6250BPI | -6250 | -1600BPI | -1600 | -800BPI | -800]
ER!
```

RELEASING A TAPE DRIVE

When a user completes a mag tape operation, the mag tape drive should be released for general use. Simply issue the UNASSIGN command with one of the indicated arguments:

$$\underline{\text{UN}}\text{ASSIGN} \left\{ \begin{array}{l} \text{MTpdn} \\ \underline{\text{-A}}\text{LIAS MTldn} \end{array} \right\}$$

The -ALIAS option can be used to unassign a drive whether or not the user assigned an alias to the drive. The ldn argument value can be either the user-chosen logical device number, if one was assigned, or the default ldn, which is identical to the pdn.

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

The system operator can unassign any drive using the pdn argument; the "-ALIAS ldn" option can be used only if the drive is owned by (i.e., was previously assigned by) the operator.

If an operator UNASSIGNs a user-dedicated tape drive, no message will appear at that user's terminal. Should the user subsequently attempt to UNASSIGN the same device, an error message will be displayed.

MAG TAPE OPERATIONS

Each magnetic tape utility performs one or more specific functions.

MAGNET (for both Prime and non-Prime-format files and tapes)

- Reading files from tape to disk (with optional unblocking or character translation)
- Writing files from disk to tape (with optional blocking or character translation)
- Copying files from one tape to another
- Translation from EBCDIC or BCD to ASCII during READ or WRITE operations (optional)
- Copying binary files

MAGRST (Prime-format tapes only)

- Restoring Prime-format files, directory-trees or disk volumes from tape

MAGSAV (Prime-format files only)

- Archiving Prime-format files, directory-trees or disk volumes to tape

The dialogs associated with these utilities are summarized below. For complete information on these utilities, see The PRIMOS Commands Reference Guide.

THE MAGNET UTILITY

The five MAGNET options perform the following tape operations:

Option	Function
COPY	Copies files from one tape to another
POSITION	Positions tape to a file or record
QUIT	Returns to PRIMOS
READ	Reads files from tape to disk
WRITE	Writes a file from disk to tape

MAGNET Requirements

Acceptable Tapes: MAGNET accepts only unlabeled tapes with fixed-length records and optional blocking. They may be 7- or 9-track, and may be written in ASCII, BCD (7-track only), BINARY or EBCDIC format. They may have a maximum of 10K bytes/tape record, and a maximum of 2K bytes/disk record.

Tapes which meet these standards may be read, written or copied with MAGNET. Translation from/to ASCII, BCD, BINARY and EBCDIC can be done during READ or WRITE operations. Record blocking/unblocking is also possible during these operations.

ANSI level 1 volume labels of certain labeled tapes can be read with the LABEL command. LABEL can also be used to write a label on an unlabeled tape. See The PRIMOS Commands Reference Guide for details.

Reading/Writing Mag Tapes: Files may be read or written (saved) to tape with the READ and WRITE options of MAGNET, respectively. Tapes created with MAGNET cannot be restored with MAGRST, so once you save files to tape with MAGNET WRITE, they must be read back with MAGNET READ. See The PRIMOS Commands Reference Guide for complete details on the MAGNET READ and WRITE options.

Copying Tapes: The COPY option allows files to be copied from one tape to another. No character translation is provided for during this operation. Tapes may also be copied in their entirety with this option, as explained below.

Reading or Writing Magnetic Tape with MAGNET

Once the tape drive has been assigned and the tape mounted, users may read tapes with the READ option of PRIMOS' MAGNET utility. When the command MAGNET is given, an interactive dialog begins. (The same dialog, with the WRITE option, allows users to write tapes.)

```
    OK, MAGNET

    [MAGNET rev 18.1]

    OPTION:  READ


    MTU# = unit-number [/tracks]
```

unit-number is the number of the magnetic tape drive unit which was previously assigned.

tracks is either 7 or 9; if this parameter is omitted, 9-track tape is assumed.

MAGNET then asks a series of questions about the tape format.

```
    MTFILE# = tape-file-number
```

tape-file-number is the file number on the tape. A positive integer causes the tape to be rewound and then positioned to the file number; a 0 causes no repositioning of the tape.

```
    LOGICAL RECORD SIZE = n
```

This is the number of bytes/line image; normally this is 80 for a source program.

BLOCKING FACTOR = blocking-factor

blocking-factor is the number of logical records per tape record. (Maximum size of a tape record is 10,000 characters.)

ASCII, BCD, BINARY, OR EBCDIC? data-representation

data-representation	action
ASCII	Transfer
BCD	Translate to ASCII from 7-track tape
BINARY	Transfer verbatim
EBCDIC	Translate to ASCII

FULL OR PARTIAL RECORD TRANSLATION? answer

answer is FULL or PARTIAL. The question is asked only for BCD or EBCDIC representations. Partial translation allows specified bytes in the record to be transferred to disk without translation to ASCII. The partial option is useful when transferring data files with binary or packed decimal EBCDIC data. However, almost all source programs will be transferred with the full option.

OUTPUT FILENAME: filename

filename is the name of the file in the UFD into which the magnetic tape is to read. If the filename already exists in the UFD, the question:

OK TO DELETE OLD filename? answer

will be asked. A NO will cause the request for an output filename to be repeated. A YES will cause the transfer to begin; upon completion, the following message will be printed out:

```
DONE, tape-records RECORDS READ, disk-records DISK RECORDS OUTPUT
OK,
```

Use of the tape drive unit should then be relinquished by UN MTpdn or UN -ALIAS ldn.

DUPLICATING MAGNETIC TAPES

MAGNET can copy and read either Prime or non-Prime tapes. MAGSAV creates Prime-format tapes which can then be read by MAGRST.

Copying Tapes with MAGNET: If there are two tape drives available for use, the COPY option of MAGNET can be used to generate duplicates of magnetic tapes. This option copies one tape directly to another. The MAGNET utility may be used for tapes in Prime or non-Prime format.

The essential steps in the copy procedure are:

1. Assign two magnetic tape drive units from terminal.
2. Mount the FROM tape (original) and TO tape (blank) on their respective drive units.
3. Use COPY option of MAGNET: supply FROM and TO tape unit numbers, starting file number and number of files to be copied, as requested by dialog (see below).
4. Dismount both tapes and unassign tape drives when EOT (end of tape) message is returned.

The MAGNET COPY Dialog: The COPY option of MAGNET invokes the following prompts. Expected user responses are outlined opposite corresponding prompts.

Prompt	Response
'FROM' TAPE: MAGNETIC TAPE UNIT NUMBER=	Enter number (ldn or pdn) of mag tape drive on which non-blank tape is mounted.
STARTING FILE NUMBER=	Enter number of file to be copied; numbers correspond to order in which files appear on tape.
'TO' TAPE MTU NUMBER=	Enter number (ldn or pdn) of mag tape drive unit on which blank tape is mounted.
STARTING FILE NUMBER=	Enter position on tape where file will reside.
NUMBER OF FILES TO COPY=	Enter number of files to be copied. If copying entire tape, enter a large number; operation ceases when EOT is reached.
DONE	This means the operation is completed. The number of files copied is printed and control returns to PRIMOS.

Copying Tapes with MAGRST/MAGSAV: When copying tapes saved with MAGSAV, the MAGSAV/MAGRST utilities can be used to duplicate tapes as follows:

1. Assign a tape drive unit from the terminal.
2. Mount FROM (original) tape on drive unit.
3. Copy tape to files on disk using MAGRST.
4. Remove FROM tape and replace the TO (blank) tape on drive unit.
5. Transfer files from disk to TO tape using MAGSAV.
6. Dismount tape and unassign drive unit from terminal.

Saving Disk Files on Tape (MAGSAV)

The Magnetic Tape Save Utility writes PRIMOS files from disk to a 7- or 9-track magnetic tape. Several options may be specified on the MAGSAV command line:

-7TRK	Uses 7-track magtape format instead of default (9-track).
-INC	Indicates incremental dump. Only files and directories with DUMPED switch set to 0 will be saved. (Default=save all).
-LONG	Sets record size to 1024 words (Default=512).
-UPDT	Indicates update. DUMPED switch is set for files and directories saved from disk to tape.
-TTY	MAGSAV takes tape unit number from terminal and all other information from current input stream.
-VAR	Allows variable-length records, up to 2048 words; overrides -LONG option. Improves speed of MAGSAV operation. If selected, the record size is printed after the REV stamp of the MAGSAV dialog.

MAGSAV Dialog Summary: The MAGSAV dialog is summarized below. Suggested user responses are indicated.

Prompt	Response
TAPE UNIT (9 TRK):	Enter physical or logical tape drive number, from 0-7. If the -7 TRK option was not specified, (9 TRK) is displayed.

ENTER LOGICAL TAPE NUMBER:	Enter number, from 1 to n, of desired logical tape (see Note, below); tape is then rewound and positioned. Specify 0 if tape is already positioned as desired.
TAPE NAME:	Specify a name or identifier for this tape; maximum of 6 characters.
DATE:	Specify date in format: mm dd yy. Default (CR) is system-supplied date.
REV. NO.:	Enter arbitrary number, or (CR).
NAME OR COMMAND:	Possible responses include:

pathname	Name of file or directory to be saved.
MFD	Saves entire disk volume.
*	Saves current directory; up to 13 (nested) levels can be saved at a time.

$A directory [ldisk]: Changes home UFD to directory. If ldisk number is not specified, only the local disk is searched for directory (default). pathnames are not supported.

$I[filename]n: Prints at terminal an index of files and directories saved from from disk to tape. Index can be written to a file if a filename is provided. n indicates number of levels in tree structure hierarchy to be included in index.

$Q	Terminates logical tape and returns to PRIMOS.
$R	Terminates logical tape, rewinds tape and returns to PRIMOS.
$INC ON OFF	Turns incremental save option on or off; same as -INC command line option, above.

Note

A "logical tape" results from single invocation of MAGSAV. It is a unique entity, with its own header, etc. It may be a portion of a physical tape, or a complete physical tape; or it may span one or more physical tapes. A single physical tape may contain several logical tapes, each of which is identified by number.

Sample MAGSAV Session: Below is an example taken from a terminal session during which a disk file (TAPE.EX) was saved on tape. If a carriage return (CR) is given in response to the DATE and REV NO prompts, as shown below, the system will supply the current date and zero rev number. Notice that a logical device number (ldn) can be supplied as a response to the "TAPE UNIT" prompt as in this example. Either a pdn or an ldn, (if one has been assigned), can be supplied.

```
    OK, AS MT1 -ALIAS MT7
    Device MT1 Assigned.
    OK, STAT DEV

    DEVICE  USRNAM  USRNUM  LDEVICE
    MT1     DOUROS    7     MT7

    OK, MAGSAV
    REV. 18.1
    TAPE UNIT (9 TRK): 7
    ENTER LOGICAL TAPE NUMBER: Ø
    TAPE NAME: MAGTAP
    DATE (MM DD YY):(CR)
    REV NO:(CR)
    NAME OR COMMAND:  TAPE.EX
    NAME OR COMMAND: $Q
    OK,
```

Restoring Files to Disk (MAGRST)

The Magnetic Tape Restore Utility restores files, directory, trees and partitions from a magnetic tape (7- or 9-track) to a disk. All information is restored to the directory to which the user is currently attached. MAGRST can read tapes of any record size, with fixed or variable length records (up to 6144 words), making it compatible with MAGSAV.

The command format is:

```
    MAGRST [-7TRK] [-TTY] (option specifies 7-track tape:  default=9)
```

MAGRST Dialog Summary: The MAGRST utility displays a series of questions and messages which are summarized below, along with appropriate responses and descriptions. If the -TTY option is specified, MAGRST takes the unit number from the terminal, but takes all other information from its current input stream. (This might be a command file or a CPL file.)

Prompt/Message	Response/Description
YOU ARE NOT ATTACHED TO AN MFD	This message is returned only if the user is not attached to an MFD.
TAPE UNIT (9 TRK):	Enter physical or logical device number; from 0-7. The (9 TRK) message is displayed if the -7 TRK option was not specified on the MAGRST command line.
(TAPE NOT AT LOAD POINT)	This message appears if the tape is not positioned to the beginning of the tape.
ENTER LOGICAL TAPE NUMBER:	If tape is divided into several logical units, enter logical tape number from 1 to _n_. Tape is positioned to specified logical tape. Enter 0 if tape is already positioned as desired. (No action is taken in this case.) See also _Note_, below.
NAME: tape-name	MAGRST displays the name of the logical tape currently positioned to; names are provided during MAGSAV dialog.
DATE (MM DD YY): tape-date	MAGRST displays date on tape was recorded. Supplied during MAGSAV.
REV NO: number	MAGRST displays arbitrary _number_ specified during MAGSAV.
REEL NO: reel-number	MAGRST displays appropriate _reel-number_ of tape.
READY TO RESTORE:	Enter one of the following options: _YES_: Restores entire tape and returns to PRIMOS. _NO_: Causes first prompt to be reissued.

$I [filename] n: Prints tape index to n levels at terminal during restore. Index can be optionally saved to indicated filename.

NW [filename][n]: Prints n level index at terminal but DOES NOT UPDATE disk because no files are restored. Optionally stores index in filename.

PARTIAL: Restores only certain files and directories. Pathnames are entered in response to "TREE NAME:" prompt.

$ A directory [ldisk]: Changes home UFD to directory. If ldisk number is not specified, local disk is searched for directory.

TREE NAME:

This prompt is returned when PARTIAL option is specified. Respond with one of the following:

pathname: Names file or directory to be restored. pathname should not include name of directory to which user was attached when saving file or directory, except when attached to an MFD. If, for example, a file, file2, was saved from UFD=TOP, and its pathname is: TOP>MID>file2, it can be restored with the pathname: "MID>file2", but NOT with the pathname: "TOP>MID>file2".

(CR): Terminates MAGRST dialog by indicating end of treename list; tape is read, and control returns to PRIMOS.

Note

A "runaway" tape condition can occur if there is only one logical tape on the currently mounted reel of tape and the user specifies a number greater than 1 in response to the LOGICAL TAPE NUMBER prompt. If this happens, MAGRST will search endlessly for the non-existent logical tape(s) and will consequently be unable to read the end-of-tape marker. The drive must be unassigned to abort the unsuccessful search.

When an unrecoverable error is encountered during an attempted MAGRST operation, an error message is displayed. Recoverable errors are logged and a total is displayed when the end of the logical or physical tape is reached.

Sample MAGRST Session: The following example represents the dialog necessary to restore a file from tape to disk. The file saved in the previous MAGSAV sample session (TAPE.EX) is used in this example also.

```
OK, MAGRST
REV. 18.1
YOU ARE NOT ATTACHED TO AN MFD
TAPE UNIT (9 TRK):0
ENTER LOGICAL TAPE NUMBER: 1
NAME: MAGTAP
DATE(MM DD YY): 08-31-79
REV NO:       0
REEL NO:       1
READY TO RESTORE: PARTIAL
TREE NAME: TAPE.EX
TREE NAME:(CR)
*** STARTING RESTORE ***
*** END LOGICAL TAPE ***
*** RESTORE COMPLETE ***
OK,
```

SECTION 13

USING PRIMENET

INTRODUCTION

Many Prime installations contain two or more processors connected in a network--a combination of communications hardware and PRIMOS software called PRIMENET. In a network, the processor to which the user terminal is connected is the local processor, while all other processors are considered remote. On a system using PRIMENET, you can:

- LOGIN to a UFD on a remote system and use that CPU for processing. (Only terminal I/O is sent across the network.)
- LOGIN to your local UFD, then ATTACH to directories on disk volumes connected to any other processor in the network, and access files in such directories. (File data is transmitted across the network; the local CPU does the processing.)
- Use a PATHNAME with a subsystem (such as the EDITOR) to access a file on a remote disk. For example:

  ```
  ED <FOREST>OAK>BRANCH5>ACORNLIST
  ```

- Use FUTIL to copy a file from a remote directory into a local directory, avoiding the overhead of frequent remote access. For example:

  ```
  OK, futil
  [FUTIL rev 18.1]
  > from <forest>oak>branch5
  > copy acornlist
  > quit

  OK,
  ```

  An overview of FUTIL is contained in Section 11.

REMOTE LOGIN

Each processor in the system is assigned a nodename during system configuration. The nodename then identifies the processor for remote logins. (Users can determine the nodenames of remote processors by using the STATUS NETWORK command, explained below.) The format for remote logins is:

```
LOGIN ufd-name [password] -ON nodename
```

If -ON nodename is omitted, an attempt is made to log into ufd-name on the local system only. If nodename is the name of the local node, the login attempt is done locally without the use of PRIMENET.

If the LOGIN command fails for any reason (e.g., Not found, insufficient access rights), the user's PRIMENET connection is broken. Input from the user's terminal is again processed by the local processor; but the user is not logged in.

On a terminal logged in to a remote processor, the command LOGOUT logs out the process, breaks the remote connection over PRIMENET, and reconnects the terminal to its local system (not logged in). The message:

```
    WAIT . . .


    DISCONNECTED FROM xxx
    OK,
```

is displayed. All input characters typed between the LOGOUT command and the response OK are discarded.

Network Status

The STATUS NETWORK command gives the names and states of all nodes in the network:

```
    OK, status net


    RING NETWORK

         NODE      STATE
        SYSA        UP
        SYSB       ****
        SYSC        UP
        SYSD        UP
        SYSE        UP
        SYSG       DOWN
        SYSH       DOWN


    OK,
```

This shows the state of a eight-node network as it would be printed for a local user on the SYSB node. The UP state means that the node is configured and functioning.

ATTACHING TO REMOTE DIRECTORIES

Attaching to a remote directory is the same as attaching to a local directory. You can give the name of the disk partition or logical disk number (determined from a STATUS DISKS display) within the ATTACH command, as in:

```
    ATTACH <SHARK>JAWS
```

Or you may give the UFD-name by itself. PRIMOS then searches each logical disk beginning with disk 0, and attaches you to the first UFD of that name it finds.

Status Disks

Users can discover the names and numbers of logical disks on remote systems by using the STATUS DISKS command. For example, suppose we wanted to attach to the UFD CORAL on node SYSC, but had forgotten the name of the disk partition on which that UFD resided. We could accomplish the ATTACH as follows:

```
    OK, STATUS DISKS


    DISK    LDEV  PDEV  SYSN
    STATS     0   3462
    FIELDS    1    460
    MISCEL    2  71063
    FOREST    3  71061
    REEFS     4    460  SYSC
    LAGOON    5    460  SYSD
    SHARK     6  12060  SYSD
    SHARK2    7  52061  SYSD
    CLOUDS   12    460  SYSE
    CLIFF1   13  12460  SYSE
    CLIFF2   14  61461  SYSE
    AERIE    15    462  SYSE
    ROCK     23  21460  SYSA
    FALCON   24  71061  SYSA
    NEST1    25    660  SYSA
    NEST2    26  10660  SYSA

    OK, ATTACH <REEFS>CORAL
    OK,
```

In the STATUS DISKS printout, DISK is the name of the logical disk, LDEV is the logical disk number, PDEV is the physical disk identifier, and SYSN is the nodename.

ACCESSING REMOTE SYSTEMS AND NETWORKS

You may connect to any system on the Public Data Network by using the NETLINK command. This means that systems other than Prime systems and software other than PRIMENET software may be accessed. Other sites or other networks as well as jobs within these other sites and networks may be accessed.

Several NETLINK commands let you use these other systems and networks. There are basic commands and advanced commands. Basic commands allow you to enter and exit the remote systems. Advanced commands allow you to:

- Transfer files across networks
- Set data transmission characteristics
- Print the status of your connection
- Connect to and use up to four different remote systems at the same time
- Specify the various fields of the connect packet when data transmission characteristics of a foreign system differ from that of Prime's.

Only NETLINK's basic usage will be presented here. For a list of all NETLINK commands and error messages see The PRIMENET Guide.

NETLINK Usage

The basic steps to using NETLINK are as follows:

1. Enter NETLINK Command Mode by issuing the

   ```
   NETLINK
   ```

   command. When Command Mode is entered, the @ prompt appears.

2. Connect to the remote system by issuing the

   ```
   C address
   ```

   command. address is either the host address assigned by the Public Data Network or a PRIMENET system name. For example, 617, 74, and NODE1 are all valid addresses.

   When a connection has been established, the message:

   ```
   address Connected
   ```

   appears.

3. Login to the system as you would normally, entering any validation codes or passwords as required.

4. Once you finish a terminal session, logout as you would normally. The message:

   ```
   address Disconnected
   ```

   appears. When a connection to a remote host has been terminated by logging out, Command Mode is re-entered and the @ appears. You may now connect to another site or return to PRIMOS.

5. To return to PRIMOS enter the

   ```
   QUIT
   ```

   command.

NETLINK Example

Below is an example of a basic terminal session. User responses are underlined.

```
OK, NETLINK
[NETLINK Version 1.Ø]

@ C NODE1
NODE1 connected

PRIMENET 18.1 NODE1
LOGIN HOBBIT

PRIMOS Version 18.1
HOBBIT (56) LOGGED IN AT 11:43 Ø91379

Enter validation code:  SHIRE
         .
         .                  continue with normal terminal session
         .
OK, LOGOUT
HOBBIT (56) LOGGED OUT AT 11:44  Ø91379
TIME USED = Ø:Ø1 Ø:ØØ Ø:Ø1

WAIT . . .

NODE1 DISCONNECTED

@ QUIT

OK,
```

SECTION 14

SUBROUTINE LIBRARIES

This section lists the subroutines available in:

- The Applications Libraries; VAPPLB (V-mode) and APPLB (R-mode)
- The Search and Sort Libraries: VSRTLI and VMSORTS (V-mode), SRTLIB and MSORTS (R-mode)
- The Operating System Library

It is meant solely as a checklist, to tell you what subroutines are available in these libraries. The PRIMOS Subroutines Reference Guide tells you how to use them. Thus, if you wanted to know whether a certain sort routine was available, you would look for it here. Having found it, you would consult The PRIMOS Subroutines Reference Guide for full details on how to call and use it.

APPLICATIONS LIBRARY

The applications library provides programmers with easy-to-use functions and service routines falling between very high-level constructs and very low-level systems routines. The applications library is located in UFD=LIB in the files APPLIB (R-mode programs) and VAPPLB (V-mode programs). All routines in VAPPLB are pure procedure and may be loaded into the shared portion of a shared procedure. The applications library should be loaded before loading the FORTRAN library.

Programs using the applications library subroutines must define the values of the keys used in these routines. This definition is performed by placing the instruction $INSERT SYSCOM>A$KEYS in each module which uses any of these subroutines.

The applications routines may be used as functions or as subroutine calls as desired. The function usage gives additional information. The type of value of the function (LOGICAL, INTEGER, etc.) is specified for each function.

The applications library subroutines may be grouped by their functions:

File System

TEMP$A, OPEN$A, OPNP$A, OPNV$A, OPVP$A, CLOS$A, RWND$A, GEND$A, TRNC$A, DELE$A, EXST$A, UNIT$A, RPOS$A, POSN$A, TSCN$A.

String Manipulation

FILL$A, NLEN$A, MCHR$A, GCHR$A, TREE$A, TYPE$A, MSTR$A, MSUB$A, CSTR$A, CSUB$A, LSTR$A, LSUB$A, JSTR$A, FSUB$A, RSTR$A, RSUB$A, SSTR$A, SSUB$A

User Query

YSNO$A, RNAM$A, RNUM$A

System Information

TIME$A, CTIM$A, DTIM$A, DATE$A, EDAT$A, DOFY$A

Conversions

ENCD$A, CNVA$A, CNVB$A, CASE$A, FDAT$A, FEDT$A, FTIM$A

Mathematical Routines

RNDI$A, RAND$A

Parsing

CMDL$A

A brief description of these routines follows, in alphabetical order.

CASE$A

Converts a character string from uppercase to lowercase or vice versa and returns .TRUE. if operation succeeds.

CLOS$A LOGICAL

Attempts to close a file by the file unit number on which it was opened. Reports on success or failure of attempt.

CMDL$A LOGICAL

Parses a PRIMOS-like command line and returns information for each -keyword (and optional argument) entry in the line (one entry is returned per call).

CNVA$A LOGICAL

Converts an ASCII digit string to a numerical value (INTEGER*4) for binary, octal, decimal, and hexadecimal numbers. Reports whether the conversion was made successfully or not.

CNVB$ INTEGER*2

Converts a number (INTEGER*4) to an ASCII digit string for binary, decimal, octal, and hexadecimal numbers. The function value is the number of digits in the string (or Ø if the conversion is unsuccessful).

CSTR$A LOGICAL

Compares two character strings for equality and returns .TRUE. as the function value if they are equal.

CSUB$A LOGICAL

Compares two substrings of character strings for equality and returns .TRUE. as the function value if they are equal.

CTIM$A REAL*8

Returns the CPU time since login in centiseconds (argument returned) and in seconds (function value).

DATE$A REAL*8

Returns the system date as DAY, MON DD 19YR (argument returned) and as MM/DD/YY (function value).

DELE$A LOGICAL

Attempts to delete a file specified by the filename. If successful the function is .TRUE., otherwise .FALSE..

DOFY$A REAL*8

Returns the day of the year as a 3-digit number (argument returned) and as YR.DDD (function value). The latter is suitable for printing in FORMAT F6.3.

DTIM$A REAL*8

Returns disk time since login in centiseconds (argument returned) and in seconds (function value).

EDAT$A REAL*8

Returns the date as DAY, DD MON 19YR (argument returned) and as DD/MM/YR (function value). This is the European/military format.

ENCD$A LOGICAL

Encodes a value in FORTRAN floating-point print format (Fw.d) and reports whether the encoding was successful or not.

EXST$A LOGICAL

Checks for the existence of a file specified by name and reports whether the file exists or not.

FDAT$A REAL*8

Converts the date-last-modified (DATMOD) field of a directory entry to DAY, MON DD YEAR (argument returned) and MM/DD/YY (function value).

FEDT$A REAL*8

Converts the date-last-modified (DATMOD) field of a directory entry to DAY, MON DD YEAR (argument returned) and MM.DD.YY (function value).

FILL$A INTEGER

Fills a character string with a specified ASCII character.

FSUB$A LOGICAL

Fills a character substring with a specified character and returns .TRUE. if successful.

FTIM$A REAL*4

Converts the time-last-modified (TIMMOD) field of a directory entry to HH:MM:SS (argument returned) and decimal hours (function value).

GCHR$A	INTEGER

Accesses a character in a specified character position. The function value is the character in FORTRAN A1 FORMAT (right padded with blanks).

GEND$A	LOGICAL

Positions a file pointer opened on a specified file unit to the End-of-File. The function value tells whether the positioning was successful or not.

JSTR$A	LOGICAL

Right-justifies or left-justifies, or centers a string and reports whether the operation is successful.

LSTR$A	LOGICAL

Locates a string within another string. The function value reports on whether the substring was found or not.

LSUB$A	LOGICAL

Locates one substring within another substring. The function value reports on whether the substring was found or not.

MCHR$A	INTEGER

Replaces a character in one array with a specified character from another. The function value is the character moved in FORTRAN A1 FORMAT, right padded with blanks.

MSTR$A	INTEGER

Moves one string to another string. The function value is equal to the number of characters moved.

MSUB$A	INTEGER

Moves a substring in a string into a substring in another string. The function value is equal to the number of characters moved.

NLEN$A	INTEGER*2

Returns the length (not including trailing blank) of string in a buffer.

OPEN$A LOGICAL

Opens a file on a user- or system- specified file unit. The function value reports whether the operation was successful or not.

OPNP$A LOGICAL

Gets a filename from the user terminal and opens that file on a specified file unit. The function value reports whether the operation was successful or not.

OPNV$A LOGICAL

Opens a file on a user- or system- specified file unit, verifies the operation. If the file is in use, the operations are retried. The function value reports on the ultimate success of the operations.

OPVP$A LOGICAL

Gets a file name from the user terminal and opens that file on a specified file unit. The operations are verified . If the file is in use the operations are re-tried. The function value reports on the ultimate success of the operations.

POSN$A LOGICAL

Positions the pointer in the file open on a specified file unit. The function value reports on the success of the operation.

RAND$A REAL*8

Updates the seed of a random number generator. The old seed is passed and a new seed returned. The function value is a uniform random number between Ø.Ø and 1.Ø.

RNAM$A LOGICAL

Prints a prompt message at the terminal and accepts a name from the terminal. The function value reports on the validity of the name.

RNDI$A REAL*8

Generates the initializing seed for a random number generator. The information returned is time of day in centiseconds (argument returned) and in seconds (function value).

RNUM$A LOGICAL

Prints a prompt message at the terminal and accepts a number (octal, decimal, or hexadecimal) string from the terminal. If successful the value is returned in one of the subroutine arguments and the function value is .TRUE..

RPOS$A LOGICAL

Returns the current absolute position of the pointer in the file opened on a specified file unit. The function value reports on the success of the operation.

RWND$A LOGICAL

Rewinds the file opened on the specified file unit. The function value reports on the success of the operation.

TEMP$A LOGICAL

Opens a temporary file with a unique, name in the current UFD for reading and writing on a user- or system- specified file unit. The name is returned as an argument in the subroutine call. The function value reports on the success of the operation.

TIME$A REAL*8

Returns the time of day as HR:MN:SC (argument returned) and in decimal hours (function value).

TREE$A LOGICAL

Scans a string to check whether it is a valid pathname and, if so, locates the final part (filename) of the name in the string. The function value reports whether the test is successful or not.

TRNC$A LOGICAL

Truncates the file opened on a specified file unit. The function value reports on the success of the operation.

TSCN$A LOGICAL

Scans the file system tree-structure (starting with the home directory) to read UFDs and segment directory entries. Each call returns the next file on the current level or the first file on the next lower level. The function value is .TRUE. until an error occurs or an end of file is reached.

TYPE$A LOGICAL

Tests a character string to see whether it can be interpreted as a number (binary octal, decimal, or hexadecimal) or a name. The function value reports whether the string meets the specified criterion.

UNIT$A LOGICAL

Tests whether any file is open on a specified file unit. The function value reports whether the unit is in use or not.

YSNO$A LOGICAL

Prints a question at the user terminal which can be answered YES (or OK) or NO. The function value is .TRUE. for YES (or OK) and .FALSE. for NO. Any other answer causes the question to be repeated.

SORT AND SEARCH LIBRARIES

There are two classes of sorting subroutines available: disk sorts and in-memory sorts. Disk sorts use the mass storage devices (disks) for working space while the in-memory sorts put working information in the user's address space. For complete details on the use of these subroutines, see The PRIMOS Subroutine Reference Guide.

Disk Sorts

Disk sort subroutines are in the VSRTLI (V-mode) and SRTLIB (R-mode) libraries. VSRTLI contains the following:

- ASCS$$ sorts or merges ASCII or binary files on any of the 12 supported key types.
- SUBSRT sorts a single input file on ASCII keys. It has a simpler calling sequence than ASCS$$.
- SRTF$S sorts from one to twenty input files into a single output file. It allows specification of both input and output file types.

- MRG1$S merges from one to eleven input files into a single output file. It allows specification of both input and output file types.

The twelve supported key types are: ASCII, single-precision integer, single-precision real, double precision real, double-precision integer, numeric ASCII with leading separate sign, numeric ASCII with trailing separate sign, packed decimal, numeric ASCII with leading embedded sign, numeric ASCII with trailing embedded sign, numeric ASCII unsigned, and ASCII with lower case letters treated as equal to upper case letters. SRTLIB contains the following:

- ASCS$$ sorts on ASCII (upper and lower case) or binary keys. It can also merge up to ten files.

- SUBSRT sorts a single input file on ASCII keys. It has a simpler calling sequence than ASCS$$.

In-memory Sorts and Binary Search

The subroutines listed here are contained in the library MSORTS (R-mode) and VMSORTS (V-mode) in the UFD LIB. A complete discussion of these subroutines will be found in The PRIMOS Subroutines Reference Guide.

See Knuth, Donald The Art of Computer Programming, vol. 3 for complete discussion of these types of sorts.

Table 13-1 lists the characteristics of these sorts.

Table 13-1. Sort Characteristics

Sort	Approximate relative running time		Comments
	Average	Maximum	
BUBBLE	N**2	-	only good for very small N
HEAP	23N*ln(N)	26N*ln(N)	inefficient for N<2000
INSERT	N**2	-	small N; very good on nearly ordered tables
QUICK	12N*ln(N)	N**2	fastest but very slow on nearly ordered tables
SHELL	N**1.25	N**1.5	good for N<2000

N is the number of entries in the table (nentry).

These routines all sort the table in increasing order with the key treated as a single, signed multiple-word integer.

RADXEX, however, treats the key as a single, unsigned multi-word (or partial word) integer. For example: If the keys were 5, -1, 10, -3, RADXEX would sort them to: 5, 10, -3, -1 The other routines would sort them to: -3, -1, 5, 10

OPERATING SYSTEM LIBRARY

These subroutines are used mainly by PRIMOS. However, a number of them useful at the applications level are described in detail here. Complete details will be found in The PRIMOS Subroutines Reference Guide.

File Access

Files are structured to be accessed in either of two ways: SAM, or Sequential Access Method, and DAM, or Direct Access Method. SAM files are the most common type of file created and processed by PRIMOS. Most files likely to be dealt with by the user are SAM files.

SAM Files: A SAM file consists of records threaded together with forward and backward pointers. Each record in the file contains a pointer to the beginning record address (BRA) of the file. The beginning record of the file contains a pointer to the file directory in which it is listed. Since records are strung together in this manner, they can only be accessed sequentially; the entire file must be searched from the beginning in order to find a record. This is time consuming when many random accesses must be done. However, SAM files are more compact and require less disk storage space than DAM files. SAM files are accessed by PRIMOS commands such as ED, etc.

DAM Files: DAM files have a multi-level index containing pointers to every record on the file. If the file is short, the record address pointers point directly to records containing data. If the file is long, these pointers reference other records containing a lower level index. Those indices in turn have pointers to records containing data.

DAM structure is more suitable to rapid, random access of data than SAM structure. Each individual record can be referenced by a unique pointer connecting the record and a pointer index at the beginning of the file. Searching the pointer index for a particular record is quicker than hunting through each entire record in sequence.

DAM files are less compact than SAM files. The MIDAS subsystem or user applications programs must be written to access them. DAM files occur in the MIDAS and SEG subsystems.

Names

In the file system calls, names are either ASCII, packed two characters per word, or character strings (the actual name preceded and followed by a single quote). If the name length specified in a call is longer than the actual length of the name, the name must be followed by a number of trailing blanks sufficient to match the given length.

Passwords

Passwords can be at most six characters long. Passwords less than six characters must be padded with blanks for the remaining characters. Passwords are not restricted by filename conventions and may contain any characters or bit patterns. It is strongly recommended that passwords not contain blanks, commas, the characters = ! ' @ { } [ ] ( ) or lowercase characters. Passwords should not start with a digit. If passwords contain any of the above characters or begin with a digit, the passwords may not be given on a PRIMOS command line to the ATTACH command.

Keys and Error Codes

All keys and error codes are specified in symbolic, rather than numeric form. These symbolic names are defined as PARAMETERS for FORTRAN programs in $INSERT files in a UFD on the master disk called SYSCOM. The key definition file is named KEYS.F for FORTRAN. The error definition file is ERRD.F.

Error Handling

Errors occurring from a subroutine call causes a non-zero value of the argument CODE to be turned. Users should always test CODE after a call for non-zero values to be certain no errors are missed. Error printing and control are performed by the ERRPR$ subroutine:

```
CALL ERRPR$ (key,code,text,text-length,name,name-length)
```

<u>key</u>	Action to be taken after printing message.
K$NRTN	Exit to PRIMOS; do not allow return to calling program.
K$SRTN	Exit to PRIMOS; return to calling program following a START command.
K$IRTN	Return immediately to calling program.
<u>code</u>	An integer variable containing the error code returned by the subroutine generating the error.

text	User's message to be printed following standard error message (up to 64 characters).
text-length	Length of text in characters. To omit text, specify both text and text-length as 0.
name	User-specified name of program or sub-system, detecting or reporting the error (up to 64 characters).
name-length	Length of name in characters. To omit name, specify both name and name-length as 0.

The message format for non-zero values of CODE is:

standard text. user's text, if any (name, if any) e.g.,

```
ILLEGAL NAME.  OPENING NEWFILE (NEWWRT)
```

These errors are included in the list of run-time errors in Appendix A. They are labeled as File System errors.

Operating System Subroutines

A list of operating system subroutines with a brief description of their functions is given below. Subroutines marked with a bullet (●) are described in detail following this list.

●	ATCH$$	Attaches to a UFD and optionally makes it the home UFD.
●	CNAM$$	Changes a filename.
	COMI$$	Switches command input stream from terminal to command file and vice-versa.
	COMO$$	Switches output stream from terminal to file and vice-versa.
	CREA$$	Creates a sub-UFD in the current UFD.
	ERKL$$	Reads or sets the erase and kill characters.
	FORCEW	Writes immediately to the disk all modified records of the file currently open on funit.
	GPAS$$	Returns passwords of sub-UFD in the current UFD.
	GPATH$	Obtains a fully qualified pathname for an open file unit.
	NAMEQ$	Compares filenames for equivalence.
●	PRWF$$	Reads, writes, and positions pointer in a SAM or DAM file.

RDEN$$	Reads entry in UFD.
RDLIN$	Reads line of characters from compressed or uncompressed ASCII disk file.
RDTK$$	Parses the command line, token by token.
REST$$	Restores an R-mode memory image to user memory from a disk file.
● RESU$$	Restores an R-mode memory image from a file, sets initial values, and begins execution. An error in this call causes an error message to be printed automatically and then returns command to PRIMOS.
SATR$$	Sets attributes (protection, date, time, etc.) in a UFD entry.
SAVE$$	Saves an R-mode memory image in user memory by writing it into a disk file.
SGDR$$	Positions and reads segment directory entries.
SPAS$$	Sets the passwords in the current UFD.
● SRCH$$	Opens or closes a file.
TEXTO$	Checks the validity of a filename.
● TSRC$$	Opens or closes a file anywhere in the PRIMOS file structure.
WTLIN$	Writes a line of ASCII characters to a disk file in compressed format.

ATCH$$

CALL ATCH$$ (ufd-name,name-length,logical-disk,password,key,code)

ufd-name	Name of UFD to be attached to (if ufd-name=K$HOME and key=0, attachment is to home UFD).
name-length	Length in characters of ufd-name (if ufd-name=K$HOME, name-length is ignored).
logical-disk	Logical disk to searched for ufd-name when key=K$IMFD.

logical-disk	Action
K$ALLD	Search all started-up logical devices.
K$CURR	Search MFD of current disk.

password	3-word array containing the owner or non-owner password of ufd-name (if attaching to home UFD, password may be Ø).
key	reference-key + set-key

reference-key

K$IMFD Attach to ufd-name in MFD on logical-disk.
K$ICUR Attach to ufd-name in current UFD.

set-key

K$SETH Set current UFD to home after attaching.

code	Returns integer-valued error code.

CNAM$$

```
CALL CNAM$$ (old-name,old-name-length,new-name,new-name-length,
           code)
```

old-name	Name of file to be changed.
old-name-length	Number of characters in old-name.
new-name	Name to be changed to.
new-name-length	Number of characters in new-name.
code	Returns integer-valued error code.

Note

CNAM$$ requires owner-rights in the current UFD.

The names of the MFD,BOOT,BADSPT, or the packname may not be changed.

PRWF$$

```
CALL PRWF$$ (read-write-key+position-key+mode,file-unit,LOC(buffer),
            number-of-words,position-value,words-transferred, code)
```

read-write-key	Action to be taken (mandatory).
K$READ	Read number-of-words from file-unit into buffer.
K$WRIT	Write number-of-words from buffer to file-unit.
K$POSN	Set current position to value at 32-bit integer in position-value.

K$TRNC	Truncate files open on file-unit at current position.
K$RPOS	Return current positions as a 32-bit integer in position-value.
position-key	Indicates positioning (optional).
K$PRER	Move file pointer of file-unit position-value words relative to current position; then perform read-write-key operation.
KPOSR	Performs read-write-key operation then move file pointer of file-unit position-value words relative to current position.
K$PREA	Move file pointer of file-unit to absolute position-value then perform read-write-key operation.
K$POSA	Perform read-write-key operation, then move pointer of file-unit to absolute position-value.

If position-key is omitted, K$PRER is used.

mode	Transfer all or convenient number of words (optional).
omitted	Read/write number-of-words.
K$CONV	Read/write convenient number of words up to number-of-words.
K$FRCW	Perform write to disk from buffer before executing next instruction in the program. Increases disk I/O time.

See The PRIMOS Subroutines Reference Guide for a discussion of "convenient".

file-unit	File unit on which the file has been opened (by SRCH$$, PRIMOS command, etc.)
buffer	Data buffer for read/write. If not needed, specify as LOC(Ø).
number-of words	number of words to be transferred (mode=Ø) or maximum number of words to be transferred (mode=K$CONV). number-of-words may range from Ø to 65535.

position-value	Relative or absolute position value (32-bit integer, INTEGER*4). If not needed, specify long-integer zero as 000000 or INTL(0).
words-transferred	The number of words actually transferred when read-write-key=K$READ; other keys leave this parameter unmodified. (INTEGER*2).
code	Returns integer-valued error code.

RESU$$

```
CALL RESU$$    (filename,name-length)
```

filename	Name of the file containing the memory image.
name-length	Number of characters in filename.

SRCH$$

```
CALL SRCH$$ (action+reference+newfile,filename,name-length,
             file-unit,file-type,code)
```

action	Action to be taken (mandatory).
K$READ	Open filename for reading on file-unit.
K$WRIT	Open filename for writing on file-unit.
K$RDWR	Open filename for reading and writing on file-unit.
K$CLOS	Close file by filename or by file-unit.
K$DELE	Delete filename.
K$EXST	Check existence of filename.
reference	Modifies action (optional).
K$IUFD	Search for filename in current UFD (this is the default).
K$ISEG	Perform the action on the file that is a segment directory entry in the directory which is open on filename.
K$CACC	Change access rights of file open on file-unit to action.
K$GETU	Open filename on an unused file-unit selected by PRIMOS. The unit number is returned in file-unit.

new-file	Specifies type of file to create if file-name does not already exist.
K$NSAM	SAM file (this is the default).
K$NDAM	DAM file.
K$NSGS	SAM segment directory.
K$NSGD	DAM segment directory.
filename	Name of the file to be opened. If reference=K$ISEG, filename is a file unit on which a segment directory is already open.
name-length	Number of characters of filename.
file-unit	File unit number on which file is to be opened or closed.
file-type	Returns type of file opened. If call does not open file, its value is unchanged. The values are integers.

Ø	SAM file
1	DAM file
2	SAM segment directory
3	DAM segment directory
4	UFD

code	Returns an integer-valued error code.

Note

A UFD may be opened only for reading.

A UFD cannot be deleted unless it is empty.

A segment directory cannot be deleted unless it is of length Ø.

TSRC$$

```
CALL TSRC$$ (action+new-file,pathname,file-unit,character-
          position,code)
```

action	Action to be taken (mandatory).
K$READ	Open pathname for reading on file-unit.
K$WRIT	Open pathname for writing on file-unit.
K$RDWR	Open pathname for reading and writing on file-unit.

K$DELE	Delete file pathname.
K$EXST	Check on existence of pathname.
new-file	Specifies type of file to create if pathname does not already exist.
K$NSAM	SAM file (this is the default).
K$NDAM	DAM file.
K$NSGS	SAM segment directory.
K$NSGD	DAM segment directory.
pathname	A specification of any file in any directory or subdirectory stored in array pathname packed two characters per word.
file-unit	PRIMOS file unit number on which the file is to be opened or deleted. The file-unit is closed before any action is taken.
character-position	A two-element integer array. word 1 of entry: the first character in the array that is part of the pathname (count starts at Ø) returns: one past the last character that was part of the pathname. word 2 - the number of characters in the pathname.
file-type	Returns type of file opened. If call does not open file, its value is unchanged. The values are integers.

Ø	SAM file
1	DAM file
2	SAM segment directory
3	DAM segment directory
4	UFD

code	returns an integer valued error code

Note

TSRC$$ always closes the file unit, then attaches to the user's home UFD before attempting any action.

Part IV
Altering the Command Environment

SECTION 15

CUSTOMIZING YOUR ENVIRONMENT

Users can customize their command environment in four ways:

- They can use the RDY command to choose the form of prompts to be displayed at their terminal during an interactive session or in a command file.
- They can use the ABBREV command to define their own abbreviations for PRIMOS commands, and to use those abbreviations during interactive sessions.
- They can define global variables that can be used at PRIMOS level and by user programs.
- They can send messages and set their terminal's ability to receive messages with the MESSAGE command.

CHANGING THE PROMPT MESSAGE

In addition to its normal OK, and ER! prompts, PRIMOS also supplies a long form of prompt message which displays the time, the amount (in seconds) of CPU time and I/O time used since the last prompt, and the user's stack level. (The stack level is only displayed if it is greater than 1; most users don't need to worry about it.)

Users can change the form of prompt message displayed at their terminal by giving the RDY command. Its format is:

```
RDY [options]
```

If given without options, the RDY command prints a single long-form prompt. If given with options, the command changes the form and/or content of the prompt as follows:

Option	Function
-LONG	Sets the terminal to the long form of prompt.
-BRIEF	Returns it to the standard "OK,".
-OFF	Suppresses prompts entirely.
-ON	Re-enables prompts to the previous level of verbosity (long or brief).

-READY_LONG xxx	Changes the text portion of the long ready message to xxx. Default at login time is OK
-READY_BRIEF xxx	Changes the text portion of the brief ready message to xxx. Default at login time is OK,
-ERROR_LONG xxx	Changes the text portion of the long error message to xxx. Default at login time is ER
-ERROR_BRIEF xxx	Changes the text portion of the brief error message to xxx. Default at login time is ER!

For example:

```
OK, RDY -LONG
OK 13:11:41  0.827  1.739
RDY -OFF
RDY -ON
OK 13:11:56  0.181  0.315
RDY BRIEF -RB Absolutely!
Absolutely!
```

CREATING AND USING ABBREVIATIONS (ABBREV)

The PRIMOS command ABBREV allows you to create your own abbreviations for use in PRIMOS command lines. Its form is:

```
ABBREV [pathname] [options]
```

To use ABBREV, you:

- Create an empty abbreviation file.
- Define abbreviations within the file.
- Invoke ABBREV to activate the file during any work session in which you want to use your abbreviations.

When an abbreviation file is activated, PRIMOS calls its abbreviation processor to scan each PRIMOS command entered from the user's terminal. The abbreviation processor checks each word against the active abbreviation file, expands all abbreviations to their full defined form, then passes on the commands to the standard command processor. You can modify your abbreviation file at any time; but you can use it only for interactive sessions. Abbreviations will not be expanded in command files. Once your abbreviation file is activated, it remains active until you give the ABBREV -OFF command or log out.

Creating an Abbreviation File

Invoke the ABBREV command with the -CREATE option, giving a pathname which names and locates the new file. For example:

```
    ABBREV  MY_UFD>MY_UFD.ABBREV  -CREATE
```

This command creates and activates an empty abbreviation file. Therefore, the file specified must not already exist.

Defining Abbreviations

Abbreviations are defined and put into the abbreviation file by the -ADD option of the ABBREV command. This option has the form:

```
    ABBREV [pathname] -ADD name value
```

where name is the abbreviation and value is the commands and/or arguments the abbreviation specifies. For example:

```
    ABBREV  -ADD JD  JOB  -DISPLAY
```

This example enters the abbreviation "JD" in the user's abbreviation file, and defines it as standing for the command "JOB" plus the option "-DISPLAY." Whenever this abbreviation file is activated during a work session at a terminal, typing "JD" at that terminal will be equivalent to typing "JOB -DISPLAY".

Note

Beware of defining abbreviations identical to PRIMOS abbreviations. The abbreviation processor will give your abbreviation precedence. Therefore, you won't be able to use the PRIMOS abbreviation while your abbreviation file is active.

Activating an Abbreviation File

```
    ABBREV pathname [-ON]
```

activates an existing abbreviation file. PRIMOS loads the abbreviation table from the specified file and checks each word typed at the terminal against the abbreviations in the file before giving it to the command processor, expanding the abbreviations it finds into their full form.

Using Variables in Abbreviations

You can define variables within an abbreviation by using numerals flanked by ABBREV's escape character, %. The symbol %1% stands for the first word following the abbreviation; %2% stands for the second word, and so on. (Currently, up to nine variable words are allowed.) This feature is particularly handy for commands naming files. For example, defining an abbreviation by the command:

```
    ABBREV -ADD  F %1% %2% -L %2%.LIST -XREF -64V -DEBUG
```

would allow the abbreviation processor to translate the command:

```
    F FTN DRAGON
```

into the command:

```
    FTN DRAGON -L DRAGON.LIST -XREF -64V -DEBUG
```

Similarly,

```
    F F77 DRAGON
```

would become:

```
    F77 DRAGON -L DRAGON.LIST -XREF -64V -DEBUG
```

Other Options: ABBREV has options for refining definitions, changing or deleting definitions, etc. Four of common use are:

Command	Function
ABBREV -OFF	Deactivates abbreviation file.
ABBREV [pathname] -ON	Reactivates file. If pathname is not supplied, previous pathname is used.
ABBREV [pathname] -DELETE abbrev-1 [...abbrev -n]	Deletes the named abbreviations from the file.
ABBREV [pathname] -LIST	Lists the contents of the file.

For a full list of options and their uses, see The PRIMOS Commands Reference Guide.

USING GLOBAL VARIABLES

Sometimes you want to define variables that can be known to, and possibly modified by, a group of programs, rather than a single program. At these times, you can use global variables. Global variables are stored in one or more files inside your UFD or subdirectory. When you activate a global variable file, all the variables it contains can be used by you, (interactively, for PRIMOS commands), by all your CPL programs, and by programs written in high-level languages. Global variables survive program termination and logouts. Once defined, they last until you delete them.

The PRIMOS commands governing variables are shown below. They are explained in greater detail later in this section. For complete details see The CPL User's Guide or The PRIMOS Commands Reference Guide.

Command	Function
DEFINE_GVAR	Creates or activates a global variable file.
SET_VAR	Defines a new variable or changes the value of an existing variable. If the variable is a global variable, places it in the active global variable file.
LIST_VAR	Lists the variables contained in an active global variable file.
DELETE_VAR	Deletes variables from an active global variable file.

Global variables are particularly useful for providing easy communication of variable values among programs, as they may be set and referenced:

- At command level
- By any of your CPL programs
- By high-level language programs

Global variables must have names that begin with dots (.). For example:

```
.SIZE, .UFD
```

At command level, global variables are defined by the SET_VAR command. Within a CPL program, they are defined by the &ARGS directive, the &SET_VAR directive, or the SET_VAR command. They are defined from high-level programs by the GV$SET routine, and referenced within high-level language programs by the GV$GET routine. See The Subroutines Reference Guide for details.

SENDING MESSAGES

The MESSAGE command is used to send or receive messages. Either users or the operator may send messages. Messages may be sent:

- From any user terminal to any other user terminal
- From any user terminal to the supervisor terminal
- From the supervisor terminal to all users
- From the supervisor terminal to a specified user
- From the supervisor terminal to another supervisor terminal on a different node on the network

User Messages

The format of a user-to-user or user-to-operator message is:

```
MESSAGE  ┌username   ┐ [-NOW]
         └-usernumber┘

   text of message
```

To send a message to the operator, omit the username argument. To send a message to another user, give either the user's username (the name under which he or she logged in) or usernumber (the physical line number of his or her terminal). To get a list of users, their usernumbers, and the line numbers of their terminals, issue the STATUS USER command.

If you send a message to a username, all users logged into that name receive the message. If you send a message to a usernumber, only the specific terminal with that number receives the message.

text of message is the single-line message to be 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. For example:

```
*** uu hh:mm
  text of message
```

uu is the username and usernumber and hh'mm is the time of day in hours and minutes. For example:

```
*** BEECH (55) 11:16
```

If the -NOW option is specified, the message is printed immediately on the receiver's terminal.

If the -NOW option is not specified, messages are stored in a buffer and printed when the receiver returns to PRIMOS command level.

Setting Receive States

Users may set the receive state of their terminal with the MESSAGE command. One of three different states may be selected to control the flow of messages:

Option	Function
MESSAGE -ACCEPT	Enables reception of all messages.
MESSAGE -DEFER	Inhibits immediate messages. (Messages sent with -NOW option will be rejected by MESSAGE. Messages sent without the -NOW option will be received when you return to command level.)
MESSAGE -REJECT	Inhibits all messages.

Deferring or rejecting messages is useful when you do not want messages to interrupt a terminal session. This can be critical in situations where you are printing the contents of a file, for example. Deferring or rejecting messages in this instance would prevent the message from being printed along with your file's contents.

You cannot send a message while in MESSAGE -REJECT mode or MESSAGE -DEFER mode, because the receiver will not be able to respond.

Querying Receive States

You may determine what a user's terminal receive state has been set to with the -STATUS option of the MESSAGE command. Issuing the command -STATUS lists user's login names, terminal numbers, and receive states.

Option	Function
MESSAGE -STATUS	Lists the receive state of all users.
MESSAGE -STATUS username	Lists the receive state of all users with the name username.
MESSAGE -STATUS usernumber	Lists the receive state of the terminal with the number usernumber.
MESSAGE -STATUS ME	Lists the receive state of your own terminal.

Error Messages

The following error messages are sent by the MESSAGE command:

    BAD MESSAGE

This usually means that a typing error was made.

    UNKNOWN ADDRESSEE

The user to whom you are trying to send a message is not logged in.

    USER NOT RECEIVING NOW

This message means one of two things:

- If you are trying to send an immediate message (M -NOW), it means that the recipient's receive state is either DEFER or REJECT.
- If you are sending a message without the -NOW option, this warning means that the recipient's receive state is REJECT.

    USER BUSY

Either the terminal buffer or the message buffer is full.

    REQUIRES -ACCEPT ENABLED

Sender must issue MESSAGE -ACCEPT before sending message.

SECTION 16

USING THE CONDITION MECHANISM

INTRODUCTION

PRIMOS has a condition mechanism which is activated when any executing process encounters certain unusual events. These events (or conditions) fall into three categories:

- Software-puzzling situations: end of file encountered while reading data, illegal addresses, etc.
- 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, etc.
- External occurrences: situations not directly controlled by the executing process, such as the use of the BREAK key from the user's terminal

More than 30 PRIMOS-defined conditions exist. Some examples are:

Condition	Definition
ACCESS_VIOLATION$	Process has attempted to read, write or execute into a segment to which it has no access for that function.
ARITH$	Arithmetic exception.
STACK_OVF$	Process has overflowed its stack segment.
QUIT$	User has hit break key on terminal.
ILLEGAL_INST$	Process has tried to execute an illegal instruction.
ENDFILE (file)	End of file encountered while reading a PL/I file.

For a complete list of these conditions, see The PRIMOS Subroutines Reference Guide.

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, COBOL, or PMA can define their own on-units within the procedures for which they're 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 procedures' 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 most likely to be encountered by the user. This on-unit prints the following message at the user's terminal, then returns the user to PRIMOS command level:

```
Error: condition "condition" raised at "address"
       [extra information]
```

The user may then take any one of the following actions:

- Give the START command. The condition mechanism will try to resume running the program from the point at which the condition was raised.

- Give the DMSTK command. This will print (at the terminal or into a file, as the user prefers) a stack dump, which traces the sequence of calls and returns by which the program reached its current state. If you are familiar with Prime machine architecture, you may find that this command gives you enough information to solve your problem. (For details, see The PRIMOS Command Reference Guide.) The user may START a program again after dumping the stack.

- Give the DBG command to invoke the source-level debugger. Then re-run the program under DBG. If the DMSTK command didn't provide enough information to solve the problem, this is probably the best course of action to take.

- Give the RLS command to release the errant program. You will remain at PRIMOS command level and can give any PRIMOS command you choose.

Note

If the system default on-unit is invoked for a process running as a phantom or batch job, the condition mechanism prints the error message into the job's command output file and then logs the process out.

ON-UNIT ACTIONS

On-units can:

- Terminate the program via a non-local GOTO, passing control back to the main program, so that it can call EXIT and return to PRIMOS level.
- Run diagnostic routines, then terminate the program (as above).
- Repair the problem which caused the error condition and have the program resume execution from the point of interrupt.
- Ignore the error condition and resume running the program.
- Transfer control to some predetermined spot in the program, possibly in a different procedure from the one which raised the error condition.
- "Continue to signal", passing control back to the condition mechanism and telling it to hunt for another on-unit.
- Print messages, then do any of the above.
- Print messages and/or run diagnostic routines, then transfer control back to the user at the terminal (as the system default on-unit does).

WRITING ON-UNITS

User-written on-units have the advantage of being tailored to the procedures for which they are written. Since on-units have the same range of action as any subroutine, they can be as elaborate or as simple as required. On-units can even turn some error conditions into advantages: "ON ENDFILE CALL some-subroutine" can be an efficient way of terminating an indefinite-length input loop.

Within any procedure, users can define on-units for as many conditions as circumstances dictate. On-units can also be defined to handle conditions not normally recognized by PRIMOS: one subroutine (created by a call to SIGNL$ or SGNL$F) signals the condition when it occurs and another (created by a call to MKONU$, MKON$P, or MKON$F) acts as on-unit.

PRIMOS provides the following subroutines for users wishing to create their own on-units:

Subroutine	Function
MKONU$	Called by a procedure when it wishes to create an on-unit.
MKON$F	An FTN-specific version of MKONU$.
MKON$P	A version of MKONU$ used with FORTRAN 77 and PL1G.
SIGNL$,SGNL$F	Called to raise a condition.
CNSIG$	Called by an on-unit to pass control back to the condition mechanism.
RVONU$,RVON$F	Called by a procedure to revert (disable) an on-unit.
MKLB$F,PL1$NL	Used in FORTRAN programs to enable on-units to perform non-local GOTO's.

Information on how to use these subroutines is given in The PRIMOS Subroutines Reference Guide.

When writing on-units, the following rules must be observed:

- On-units can hand on control in one of three ways: by calling another procedure, by a local or non-local GOTO, or by returning to the calling procedure. (They may not call EXIT, though they may GO TO a point in the main program which does so.)
- They may set error codes as return parameters, print error messages, or signal other error conditions. But they may not call ERRRTN or use ERRPR$ with any but the immediate-return key (K$IRTN).
- Programs containing on-units must be compiled in V-mode or I-mode.

SCOPE OF ON-UNITS

On-units are usually defined at the beginning of a program or subroutine; but they may be defined at any point within the program. When the program reaches the point at which the on-unit is defined, (i.e., a call to MKONU$, MKON$P, or MKON$F) the on-unit is said to be set. However, the on-unit does not execute at this point. It does not execute unless the condition to which it responds is raised. An on-unit remains set until one of three things happens:

- The procedure within which the on-unit was defined returns (ends).
- A new on-unit for the condition is defined.
- The on-unit is reverted (disabled) by a call to RVONU$ or RVON$F.

Thus, if an on-unit for the condition ARITH$ is defined at the beginning of a program, it remains in effect throughout the program, unless it is reverted or some other on-unit for ARITH$ is defined later in the program. If a subroutine within that program defines its own on-unit for ARITH$, then that on-unit takes precedence (but only while the subroutine is executing). Each call to the subroutine re-establishes its on-unit; each return from the subroutine reverts the new on-unit and re-establishes the on-unit defined in the main program. (If no on-unit is defined within the main program, then PRIMOS' on-units are in effect when the main program is running.)

A FORTRAN EXAMPLE

Suppose you have a program which contains a subroutine called UPDATE that periodically updates journal entries, headers, etc. Once this subroutine is started, you want it to finish; a QUIT in the middle could foul up your bookkeeping. Write a subroutine called NOQUIT which responds to QUITs by printing a message at the terminal but otherwise ignoring the QUIT:

```
      SUBROUTINE NOQUIT (CP)  /*This will be the on-unit
      INTEGER*4  CP           /*CP=pointer to condition frame for QUIT$
C
      COMMON/COM/NAME   /*A variable used by UPDATE
      CALL TNOU ('Sorry, quits not allowed during update', 38)
      CALL TNOUA ('Currently processing record ', 28)
      CALL TNOU (NAME, 6)
      RETURN   /*Return to UPDATE at point where quit occurred
      END
```

Define NOQUIT as an external procedure within subroutine UPDATE, and establish it as an on-unit via the subroutine MKON$F. Note that if UPDATE (or any subroutine that calls MKONU$ or MKON$F) is to be compiled with the FTN compiler, it must obey the following rules:

- It must include a STACK HEADER 34 specification.
- It must be compiled using the -SPO option. This option allows allocation of the stack header space needed by the on-unit, but suppresses some error testing. Therefore, we advise that you first compile the on-units without the -SPO option, in order to test for coding errors that -SPO would ignore, before doing the actual compilation with -SPO.

- Since the -SPO option activates the DCLVAR option, the subroutine may not contain undeclared variables.
- It must not contain common blocks with names of five letters followed by a dollar sign (xxxxx$).

```
      SUBROUTINE UPDATE
C
     EXTERNAL NOQUIT
     STACK HEADER 34        /*Provides stack space for on-unit
     COMMON/COM/NAME        /*On-unit reports the value of this
                                /*variable
     INTEGER*2 NAME
     CALL MKON$F ('QUIT$', 5, NOQUIT)
                            /*Parameters are:
                            /*   condition-name (defined by PRIMOS)
                            /*   length of condition name
                            /*   name of on-unit subroutine
C
C
C    ...body of subroutine would go here...
C
C
C
     RETURN  /*At this point, NOQUIT's authority ceases.
     END
```

Appendices

APPENDIX A

GLOSSARY OF PRIME CONCEPTS AND CONVENTIONS

The following is a glossary of concepts and conventions basic to Prime computers, the PRIMOS operating system, and the file system.

- binary file

A translation of a source file generated by a language translator (FTN, PL1G, F77, COBOL, PMA, RPG). Such files are in the format required as input to the loaders. Also called "object file".

- byte

8 bits; 1 ASCII character.

- condition mechanism

A PRIMOS facility which responds to conditions that would normally cause program termination. Rather than terminating the program immediately, the condition mechanism activates an on-unit to take some diagnostic or remedial action. A list of conditions handled by PRIMOS' condition mechanism is given in the Subroutine Reference Guide.

- CPU

Central Processor Unit (the Prime computer proper as distinct from peripheral devices or main memory).

- current directory

A temporary working directory explained in the discussion on Home vs Current Directories in Section 2.

- directory

A file directory; a special kind of file containing a list of filenames and/or other directories, along with information on their characteristics and location. MFDs, UFDs, and subdirectories (sub-UFDs) are all directories. (Also see segment directory.)

- directory name

The file name of a directory.

- external command

A PRIMOS command existing as a runfile in the command directory (CMDNC0). It is invoked by name, and executes in user address space. No system-wide abbreviations exist for external commands. Users may define abbreviations for external commands by using the ABBREV command.

- file

An organized collection of information stored on a disk (or a peripheral storage medium such as tape). Each file has an identifying label called a filename.

- filename

A sequence of 32 or fewer characters which names a file or a directory. Within any directory, each filename is unique. Directory names and a filename may be combined into a pathname. Most commands accept a pathname wherever a filename is required.

Filenames may contain only the following characters:

A-Z, 0-9, _ # $ - . * &

The first character of a filename must not be numeric. On some devices underscore (_) prints as backarrow (<-).

- filename conventions

Suffixes indicate various types of files. (A dot separates the suffix from the base name of the file.) The conventions are:

filename.compiler-name	Source file (see list in Table 7-1)
filename.LIST	Listing file
filename.BIN	Binary (object) file
filename.MAP	Load map file
filename.SEG	V-mode runfile (executable)
filename.SAVE	R-mode runfile (executable)
filename.CPL	CPL file
filename.PH	Phanton command file
filename.CO	Command input file
filename.COMO	Command output file

A previous convention, which used prefixes to designate file types, is still supported for compatability. Filenames under this convention are as follows:

B_filename	Binary (Object) file
C_filename	Command input file
L_filename	Listing file
M_filename	Load map file
O_filename	Command.poutput file
PH_filename	Phantom command file
filename	Source file or text file
*filename	SAVED.p(Executable) R-mode runfile
#filename	SAVED (Executable) V-mode runfile

- file-unit

A number between 0 and 127 ('177, or octal 177)) assigned as a pseudonym to each open file by PRIMOS. This number may be given in place of a filename in certain commands, such as CLOSE. PRIMOS-level internal commands require octal values. Each user is guaranteed at least 16 file units at a time. The maximum number of units that a user may have open simultaneously varies per installation; the default is 128. PRIMOS always reserves units 0 and 127 for its own use.

- file protection keys

See keys, file protection.

- home directory

The user's main working directory, initially the login directory. A different directory may be selected with the ATTACH command.

- identity

The addressing mode plus its associated repertoire of computer instructions. Programs compiled in 32R or 64R mode execute in the R-identity; programs compiled in 64V mode execute in the V-identity. Programs compiled in 32I mode execute in the I-identity. R-identity, V-identity and I-identity are also called R-mode, V-mode, and I-mode.

- internal command

A command that executes in PRIMOS address space. Does not overwrite the user memory image. PRIMOS-defined abbreviations exist for internal commands.

- keys, file protection

Specify file protection, as in the PROTEC command.

0	No access
1	Read
2	Write
3	Read/Write
4	Delete and truncate
5	Delete, truncate and read
6	Delete, truncate and write
7	All rights

- LDEV

Logical disk device number as printed by the command STATUS DISKS. (See ldisk.)

- ldisk

A parameter to be replaced by the logical unit number (octal) of a disk volume. It is determined when the disk is brought up by a STARTUP or ADDISK command. Printed as LDEV by STATUS DISKS.

- logical disk

A disk volume that has been assigned a logical disk number by the operator or during system startup.

- MFD

The Master File Directory. A special directory that contains the names of the UFDs on a particular disk or partition. There is one MFD for each logical disk.

- mode

An addressing scheme. The mode used determines the construction of the computer instructions by a compiler or assembler. (See identity.)

- nodename

Name of system on a network; assigned when local PRIMOS system is built or configured.

- number representations

xxxxx	Decimal
'xxxxx	Octal
$xxxxx	Hexadecimal

- object file

See binary file.

- on-unit

A begin block (in PL/I) or subroutine (in FORTRAN, COBOL, or PL/I) which is activated by the condition mechanism to handle error conditions. PRIMOS has on-units for all conditions it recognizes. Users may also define on-units within any procedure they write. User-written on-units take precedence over system ones.

- open

Active state of a file-unit. A command or program opens a file-unit in order to read or write it.

- output stream

Output from the computer that would usually be printed at a terminal during command execution, but which is also written to a file if COMOUTPUT command was given.

- packname

See volume-name.

- page

A block of 1024 16-bit words within a segment (512 words on Prime 300).

- partition

A portion [or all] of a multihead disk pack. Each partition is treated by PRIMOS as a separate physical device. Partitions are an integral number of heads in size, offset an even number of heads from the first head. A volume occupies a partition, and a "partition of a disk" and a "volume of files" are actually the same thing.

- pathname

A multi-part name which uniquely specifies a particular file (or directory) within a file system tree. A pathname (also called treename) gives a path from the disk volume, through directory and subdirectories, to a particular file or directory. See the discussion on Pathnames in Section 2.

- PDEV

Physical disk unit number as printed by STATUS DISKS. (See pdisk.)

- pdisk

A parameter to be replaced by a physical disk unit number. Needed only for operator commands.

- phantom user

A process running independently of a terminal, under the control of a CPL program or a command file.

- procedure

In FORTRAN, a subroutine or function. In PL/I, any subroutine, function, or program. (In PL/I, procedures may contain other procedures.) In COBOL, the term usually refers to one or more related paragraphs or sections within the Procedure Division. Procedures direct the computer to perform a particular operation or a series of operations.

- process

A particular program running in a particular address space.

- reserved characters

The following characters are reserved by PRIMOS for special uses. They may not be used in file names.

```
( ) ` [ ] ! { } ^ " ? : ~ | < > @ + ' % \ (delete or rubout)
```

- runfile

Executable version of a program, consisting of the loaded binary file, subroutines and library entries used by the program, COMMON areas, initial settings, etc. (Created using LOAD or SEG.)

- SEG

Prime's segmented loading utility.

- segment

A 65,536-word block of address space.

- segment directory

A special form of directory used in direct-access file operations. Not to be confused with directory, which means "file directory".

- segno

Segment number.

- source file

A file containing programming language statements in the format required by the appropriate compiler or assembler.

- subdirectory

A directory that is in a UFD or another subdirectory.

- sub-UFD

Same as subdirectory.

- treename

A synonym for pathname.

- UFD

A User File Directory, one of the Directories listed in the MFD of a volume. It may be used as a LOGIN name.

- unit

See file-unit.

- volume

A self-sufficient unit of disk storage, including an MFD, a disk record availability table, and associated files and directories. A volume may occupy a complete disk pack or be a partition within a multi-head disk pack.

- volume-name

A sequence of 6 or fewer characters labeling a volume. The name is assigned during formatting (by MAKE). The STATUS DISKS command uses this name in its DISK column to identify the disk.

- word

As a unit of address space, two bytes or 16 bits.

APPENDIX B

SYSTEM DEFAULTS AND CONSTANTS

```
TERMINAL
   full duplex
   X-ON/X-OFF disabled


EDITOR (ED)
   INPUT (TTY)
   LINESZ 144
   MODE NCKPAR
   MODE NCOLUMN
   MODE NCOUNT
   MODE NNUMBER
   MODE NPROMPT
   MODE PRALL
   VERIFY
Symbols
   BLANKS  #
   COUNTER @
   CPROMPT $
   DPROMPT &
   ERASE   "
   ESCAPE  ^
   KILL    ?
   SEMICO  ;   end of line or command
   TAB     \
   WILD    !


SEGMENTED-LOADER (SEG)
   Loading address:  current TOP+1 in
         current procedure segment
   Stack size:  '6000 words
   Library:  PFTNLB and IFTNLB libraries


VIRTUAL LOADER (LOAD)
   Memory Location:  '122770 - '144000
   Loading address:  current *PBRK value
   Library:  FTNLIB  FORTRAN library
   MODE: D32R
   Sector Zero Base Area:
     Base start at location '200
     Base range '600 words
   COMMON:  Top = '077777
```

EXECUTION

A-register value	0
B-register value	0
X-register value	0
Program start address	'1000

Bits 4-6 of Keys:

000	16K, sector-address
001	32K, sector-address
010	64K, relative-address
011	32K, relative-address
110	64K, segmented-address

PRIMOS

ERASE	"
INTERRUPT	CONTROL-P or BREAK
KILL	?

Files:
- created with protection
  - owner all access rights (7)
  - non-owner no access rights (0)

APPENDIX C

ASCII CHARACTER SET

The standard character set used by Prime is the ANSI, ASCII 7-bit set, shown in Tables C-1 and C-2. This character set conforms to ANSI X3.4-1968. (1963 variances are noted.)

PRIME USAGE

Prime hardware and software uses standard ASCII for communications with devices. The following points are particularly important to Prime usage.

- Output Parity is normally transmitted as a zero (space) unless the device requires otherwise, in which case software will compute transmitted parity. Some controllers (e.g., MLC) may have hardware to assist in parity generations.
- Input Parity is ignored by hardware and by standard software. Input drivers are responsible for making the parity bit suit the host software requirements. Some controllers (e.g., MLC) may assist in parity error detection.
- The Prime internal standard for the parity bit is one, i.e., '200 is added to the octal value.

KEYBOARD INPUT

Non-printing characters may be entered into text with the logical escape character ^ and the octal value. The character is interpreted by output devices according to their hardware. (For example, typing ^207 will enter one character into the text.)

CTRL-P	('220)	is interpreted as a .BREAK.
.CR.	('215)	is interpreted as a newline (.NL.)
"	('242)	is interpreted as a character erase
?	('277)	is interpreted as line kill
\	('334)	is interpreted as a logical tab (Editor)

Table C-1. ASCII Character Set (Non-Printing)

Octal Value	ASCII Char	Comments/Prime Usage	Control Char
200	NULL	Null character - filler	^@
201	SOH	Start of header (communications)	^A
202	STX	Start of text (communications)	^B
203	ETX	End of text communications	^C
204	EOT	End of transmission (communications)	^D
205	ENQ	End of I.D. (communications)	^E
206	ACK	Acknowledge affirmative (communications)	^F
207	BEL	Audible alarm (bell)	^G
210	BS	Back space one position (carriage control)	^H
211	HT	Physical horizontal tab	^I
212	LF	Line feed; ignored as terminal input	^J
213	VT	Physical vertical tab (carriage control)	^K
214	FF	Form feed (carriage control)	^L
215	CR	Carriage return (carriage control) (1)	^M
216	SO	RRS-red ribbon shift	^N
217	SI	BRS-black ribboon shift	^O
220	DLE	RCP-relative copy (2)	^P
221	DC1	RHT-relative horizontal tab (3)	^Q
222	DC2	HLF-half line feed forward (carriage control)	^R
223	DC3	RVT-relative vertical tab (4)	^S
224	DC4	HLR-half line feed reverse (carriage control)	^T
225	NAK	Negative acknowledgement (communications)	^U
226	SYN	Synchronocity (communications)	^V
227	ETB	End of transmission block (communications)	^W
230	CAN	Cancel	^X
231	EM	End of Medium	^Y
232	SUB	Substitute	^Z
233	ESC	Escape	^[
234	FS	File separator	^\
235	GS	Group separator	^]
236	RS	Record separator	^^
237	US	Unit separator	^_

Notes

1. Interpreted as .NL. at the terminal.

2. .BREAK. at terminal. Relative copy in file; next byte specifies number of bytes to copy from corresponding position of preceeding line.

3. Next byte specifies number of spaces to insert.

4. Next byte specifies number of lines to insert.

Table C-2. ASCII Character Set (Printing)

Octal Value	ASCII Character	OCTAL Value	ASCII CHaracter	OCTAL Value	ASCII Character
24Ø	.SP (1)	3ØØ	@	34Ø	` (9)
241	!	3Ø1	A	341	a
242	" (2)	3Ø2	B	342	b
243	# (3)	3Ø3	C	343	c
244	$	3Ø4	D	344	d
245	%	3Ø5	E	345	e
246	&	3Ø6	F	346	f
247	' (4)	3Ø7	G	347	g
25Ø	(	31Ø	H	35Ø	h
251	)	311	I	351	i
252	*	312	J	352	j
253	+	313	K	353	k
254	, (5)	314	L	354	l
255	-	315	M	355	m
256	.	316	N	356	n
257	/	317	O	357	o
26Ø	Ø	32Ø	P	36Ø	p
261	1	321	Q	361	q
262	2	322	R	362	r
263	3	323	S	363	s
264	4	324	T	364	t
265	5	325	U	365	u
266	6	326	V	366	v
267	7	327	W	367	w
27Ø	8	33Ø	X	37Ø	x
271	9	331	Y	371	y
272	:	332	Z	372	z
273	;	333	[	373	{
274	<	334	\	374	|
275	=	335	]	375	}
276	>	336	^ (7)	376	~ (1Ø)
277	? (6)	337	_ (8)	377	DEL (11)

Notes

1. Space forward one position
2. Terminal usage - erase previous character
3. £ in British use
4. Apostrophe/single quote
5. Comma
6. Terminal usage - kill line
7. 1963 standard ↑; terminal use - logical escape
8. 1963 standard ←
9. Grave
10. 1963 standard ESC
11. Rubout - ignored

APPENDIX D

ERROR MESSAGES

INTRODUCTION

Error messages are given in the following order:

SEG Loader Error Messages
Loader Error Messages
Run-Time Error Messages
Batch Error Messages and Warnings

In each group errors are listed alphabetically.

Run-time error messages beginning with a filename, device name, UFDname, etc., are alphabetized according to the first word which is constant. The user should have no trouble in determining this word (the second word in the message). Leading asterisks, etc., are ignored in alphabetizing. All run-time errors have been grouped together to facilitate lookup by the user.

SEG LOADER ERROR MESSAGES

BAD OBJECT FILE

User is attempting to load file which has faulty code. The file may not be an object file or it may be incorrectly compiled. Fatal error, the load must be aborted.

CAN'T LOAD IN SECTORED MODE

The Loader is attempting to load code in sectored mode which has not been compiled in sectored mode. This could arise if trying to load a module compiled or assembled in 16S or 32S mode. It is unlikely that the average applications programmer will encounter this. Fatal error, abort load.

CAN'T LOAD IN 64V OR 64R MODE

The Loader is attempting to load code in 64V mode which is not compiled in that mode. This would arise if:

1. A program was compiled in a mode other than 64V.

2. A PMA module is written in code other than 64V and its mode is not specified.

In case 1, the user should recompile the program.

In case 2, which the average applications programmer is unlikely to encounter, the PMA module must be modified. Fatal error, abort load.

COMMAND ERROR

An unrecognized command was entered or the filenames/parameters following the command are incorrect. Usually not fatal.

EXTERNAL MEMORY REFERENCE TO ILLEGAL SEGMENT

An attempt was made to load a 64R mode program, causing a reference to an illegal segment number. Recompile in 64V mode. Fatal error, abort load.

ILLEGAL SPLIT ADDRESS

Incorrect use of the Loader's SPLIT command. Segments may be split at '4000 boundaries only (i.e., '4000, '10000, '14000, etc.) Not fatal; resplit segment.

MEMORY REFERENCE TO COMMON IN ILLEGAL SEGMENT

An attempt was made to load a 64R mode progran wherein COMMON would be allocated to an illegal segment number. Recompile in 64V mode. Fatal error, abort load.

NO FREE SEGMENTS TO ASSIGN

All SEG's segments have been allocated; no more are available at present. Use SYMBOL command to eliminate COMMON from assigned segments, thus reducing the number of assigned segments required. (User may need larger version of SEG and PRIMOS). Fatal error, abort load.

NO ROOM IN SYMBOL TABLE

Unlikely to occur; no user solution. A new issue of SEG with a bigger symbol table is required. Check with analyst. As a temporary measure, user may try to reduce number of symbols used in program. Fatal error, abort load.

REFERENCE TO UNDEFINED SEGMENT

Almost always caused by improper use of the SYMBOL command to allocate initialized COMMON. Initialized COMMON cannot be located with the SYMBOL command; use R/SYMBOL or A/SYMBOL instead.

SECTOR ZERO BASE AREA FULL

Extremely unlikely to occur. Not correctable at applications level. Check with analyst. Fatal error, abort load.

SEGMENT WRAP AROUND TO ZERO

An attempt has been made to load a 64R mode program. The program has exceeded 64K and is trying to be loaded over code previously loaded. Recompile in 64V mode. Fatal error, abort load.

LOADER ERROR MESSAGES

ALREADY EXISTS !

An attempt is being made to define a new symbol; however, the symbol name is already a defined symbol in the symbol table.

BAD OBJECT FILE

The object text is not recognizable. This usually occurs when an attempt is made to load source code or when the object text was compiled or assembled for segmented loading.

BASE SECTOR 0 FULL

All locations in the sector zero base area have been used. Use the AU command to generate base areas at regular intervals, or use the SETB or LOAD commands to specifically place base areas.

CAN'T DEFER COMMON, OLD OBJECT TEXT

The Defer Common command has been given and a module created with a pre-Rev.14 compiler or assembler has been encountered. It is not possible to defer Common in this case. The module must be recreated with a Rev.15 compiler or assembler.

CAN'T - PLEASE SAVE

The EXecute command has been given for a run file which has required virtual loading. SAve the runfile and give the EXecute command.

CM$

Command line error. Unrecognized command given. Not fatal.

COMMON OUT OF REACH

Common above '100000 is out of reach of the current load mode (16S, 32S or 32R). Use the MOde command to set the load mode to 64R.

COMMON TOO LARGE

Definition of this common block causes common to wrap around through zero. Moving the top of common - with the COmmon command - may help.

sname ILLEGAL COMMON REDEFINITION

An attempt is being made to redefine Common block sname to a longer length. The user's program should be examined for consistent Common definitions. At the very least the longest definition for a Common block should be first.

xxxxxx MULTIPLE INDIRECT

A module loading in 64R mode requires a second level of indirection at location xxxxxx. This message usually results when an attempt is made to load code compiled or assembled for 32R mode in 64R mode. It can also happen if code has accidentally been loaded into base areas as the result of a bad load command sequence.

sname xxxxxx NEED SECTOR ZERO LINK

At location xxxxxx a link is required for desectoring the instruction. No base areas are within reach except sector zero. The last referenced symbol was sname. This message is only generated when the SZ command has been given. Sname may be the name of a Common block, the name of the routine to which the link should be made, or the name of the module being loaded.

xxxxxx NO POST BASE AREA, OLD OBJECT TEXT

A post base area has been specified for module which was created with a pre-Rev.14 compiler or assembler. No base area is created. Recreate the object text with a Rev.15 compiler or assembler. This is not a fatal error.

PROGRAM-COMMON OVERLAP

The module being loaded is attempting to load code into an area reserved for Common. Use the loader's COmmon command to move Common up higher.

PROGRAM TOO LARGE

The program has loaded into the last location in memory and has wrapped around to load in Location 0. The program size must be decreased. Alternatively, compile in 64V mode and use SEG.

REFERENCE TO UNDEFINED COMMON

An attempt is being made to link to a Common name which has not been defined. This usually happens to users creating their own translators.

SECTORED LOAD MODE INVALID

A module compiled or assembled to load in R mode has been loaded in S mode. Use the MOde command to reset the load mode. It might be a good idea to be sure that all modules are correctly written, since the default load mode is 32R.

SYMBOL NOT FOUND

An attempt is being made to equate two symbols with the SYmbol command and the old symbol does not exist.

SYMBOL TABLE FULL

The symbol table has expanded down to location '4000. The last buffer cannot be assigned to the symbol table. Rebuild LOAD to load in higher memory locations, or reduce the number of symbols in the load.

SYMBOL UNDEFINED

An attempt is being made to equate two symbols; however, the old symbol is an undefined symbol in the symbol table.

64R LOAD MODE INVALID

A module compiled or assembled to run in only 32K of memory is being loaded in 64R mode. Recompile or reassemble or change the load mode with the loader's MOde command.

RUN-TIME ERROR MESSAGES

ACCESS VIOLATION — 64V mode

Attempt to perform operations in segments to which user has no right.

****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. [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 1Ø - ARGUMENT <=Ø — V-mode function

Argument not greater than zero used in logarithm (ALOG, ALOG 1Ø) function.

filename ALREADY EXISTS — Old file call

Attempt to create a file or UFD with the name of one already existing. [CZ]

Already exists. — File System

Attempt made to create, in the UFD, a sub-UFD with the same name as one already existing. (CREA$$) [E$EXST]

****AT — R-mode function

Both arguments are zero in the ATAN2 function.

**** ATAN2 - BOTH ARGUMENTS = Ø — 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, e.g., incorrect parameter. [SA]

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 key in call. File System

Incorrect key value specified in subroutine argument. (ATCH$$, RDEN$$, SATR$$, SRCH$$, SGDR$$) [E$BKEY]

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 Old file call

Incorrect password specified in ATTACH subroutine. Returns to PRIMOS level attached to no UFD. [AN]

Bad password. File System

Incorrect password specified in ATCH$$ subroutine. Returns to PRIMOS level attached to no UFD. [ATCH$$] [E$BPAS]

Note

To protect UFD privacy the system does not allow the user to trap BAD PASSWORD errors.

BAD RTNREC PRIMOS

System error.

Bad segment dir unit. File System

Error generated in accessing segment directory, i.e., PRIMOS file unit specified is not a segment directory. (SRCH$$) [E$BSUN]

Bad stack format. PRIMOS
Bad stack format signalling.

Condition mechanism cannot perform requested action because the command processor stack has been damaged (system error). User is 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 dir. File System

Error encountered in truncating segment directory. (SGDR$$) [E$BTRAN]

Bad unit number. File System

PRIMOS file unit number specified is invalid - outside legal range. (PRWF$$, RDEN$$, SRCH$$, SGDR$$). [E$BUNT]

Bad use of exit. PRIMOS

The condition mechanism sends this fatal message. User is returned to PRIMOS command level. [E$NEXP]

Beginning of file. File System

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 accomodate entry to be read into it. (RDEN$$) [E$BFTS]

Command line truncated. PRIMOS

An illegal command line has been received. The command is not executed, and the user is returned to PRMOS command level. [E$TRCL]

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

Attempt was made to ASSIGN a device currently assigned to another user. [E$DVIU]

Device not assigned. File System

Attempt was made to perform I/O operations on a device before assigning that device. [E$NASS]

Device is not started. File System

Attempt was made to access a disk not physically or logically connected to the system. If disk must be accessed, systems manager must start it up. [E$DNS]

**** DEXP - ARGUMENT TOO LARGE V-mode function

The argument of the DEXP function is too large; i.e., 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

UFD has become corrupted. (ATCH$$, CREA$$, GPAS$$, RDEN$$, SATR$$, SRCH$$) [E$BUFD]. Calls to RDEN$$ return this as a trappable error; other commands return to the PRIMOS command level.

The directory is not empty. File System

Attempt was made to delete a non-empty directory. (SRCH$$) [E$DNTE]

DISK FULL Old file call

No more room for creating/extending any type of file on disk. [DJ]

The disk is full. File System

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 internal PRIMOS commands ATTACH, LISTF, and DELETE to remove files which are no longer needed.

Disk I*O Error — File System

A read/write error was encountered in accessing disk. Returns immediately to PRIMOS level. Not correctable by applications programmer. (ATCH$$, CREA$$, GPAS$$, PRWF$$, RDEN$$, SATR$$, SRCH$$, SGDR$$). [E$DISK]

Disk is write-protected. — File System

An attempt has been made to write to a disk which is WRITE-protected. [E$WTPR]

DK ERROR — Old file call

A read/write error was encountered in accessing disk. [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 the file. (PRWF$$, RDEN$$, SGDR$$) [E$EOF]

****EQ R-mode function

Exponent overflow. (A$81)

****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, i.e., 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

Attempt made to open a file already opened or to close/delete a file opened by another user, etc. (SRCH$$) [E$FDEL]

Note

At rev 18, 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.

FILE OPEN ON DELETE File System

Attempt made to delete a file which is open. (SRCH$$) [E$FDEL]

Note

At rev 18, 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.

The file is too long. File System

Attempt made to increase size of segment directory beyond size limit. (SGDR$$) [E$FITB]

****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$FLEX - DOUBLE-PRECISION DIVIDE BY ZERO 64V mode

Attempt has been made to divide by zero.

**** F$FLEX - DOUBLE-PRECISION EXPONENT OVERFLOW 64V mode

Exponent of a double-precision number has exceeded maximum.

**** F$FLEX - REAL => INTEGER CONVERSION ERROR 64V mode

Magnitude of real number too great for integer conversion.

**** F$FLEX - SINGLE-PRECISION DIVIDE BY ZERO 64V mode

Attempt has been made to divide by zero.

**** F$FLEX - SINGLE-PRECISION EXPONENT OVERFLOW 64V mode

Exponent of a single-precision number has exceeded maximum.

**** 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.

****II R-mode function

Exponentiation exceeds integer size. (E$11)

ILLEGAL INSTRUCTION AT octal-location R mode and 64V mode

An instruction at octal-location cannot be identified by the computer.

Illegal name. File System

Illegal name specified for a file or UFD. (CREA$$, SRCH$$) [E$BNAM]

Illegal remote reference. File System

Attempt to perform network operations by user not on network. [E$IREM]

ILLEGAL SEGNO 64V mode

Program referenced a non-existent segment or a segment number greater than those available to the user.

Illegal treename. File System

The string specified for a treename 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, etc. [SI]

Insufficient access rights. File System

User does not have access right to file, or does not have write access in a UFD when attempting to create a sub-UFD. (CREA$$, GPAS$$, SATR$$, SRCH$$, SGDR$$) [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>

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 would still be considered closed for writing.)

**** I**I - ARGUMENT ERROR   V-mode function

Exponentiation exceeds integer size.

****LG   R-mode function

Argument not greater than zero in ALOG or ALOG10 function.

Max number of users exceeded.   PRIMOS

The maximum allowable number of users are already using the system. (This may mean that the operator has used the MAXUSR command to decrease the number of users temporarily.)

Max 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]

NO AVAILABLE SEGMENTS   64V mode

Additional segment(s) required - none available. User should log out to release assigned segments and try again later.

No phantoms are available.   File system

An attempt has been made to spawn a phantom. All configured phantoms are already in use. [E$NPHA]

No on-unit found. Condition mechanism

Condition mechanism cannot take action. User is returned to PRIMOS command level. [D$NOON]

No room. File System

An attempt has been made to add to a table of assignable devices with a DISKS or ASSIGN AMLC command and the table is already filled. [E$ROOM]

No timer. File System

Clock not started. System error. [E$NTIM]

NO UFD ATTACHED Old file call

User not attached to a UFD [AL, SL]. Usually occurs after attempt to attach with a bad password.

No UFD attached. File System

User not attached to a UFD. (ATCH$$, CREA$$, GPAS$$, SATR$$, SRCH$$). [E$NATT] Usually occurs after attempt to attach with a bad password.

NO VECTOR R and 64V mode

User error in program has caused PRIMOS to attempt to access an unloaded element.

1. A UII, PSU, or FLEX to location 0
2. Trap to location 0
3. SVC switch on, SVC trap and location '65 is 0.

Not a segment directory. File System

Attempt to perform segment directory operations on a file which is not a segment directory. (SRCH$$) [E$NTSD]

NOT A UFD. Old file call

Attempt to perform UFD operations on a file which is not a UFD. [AR]

Not a UFD File System

Attempt to perform UFD operations on a file which is not a UFD. (ATCH$$, GPAS$$, SRCH$$). [E$NTUD]

device-name NOT ASSIGNED PRIMOS

User program has attempted to access an I/O device which 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]

filename NOT FOUND File System

File specified in subroutine call not found. (ATCH$$, GPAS$$, SATR$$, SRCH$$) [E$FNTF]

filename not found in segment dir. File System

Filename specified in subroutine call not found in specified segment directory. (SRCH$$, SGDR$$) [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).

OLD PARTITION File System

Attempt to perform, in an old file partition, an operation possible only in a new file partition; e.g., date/time information access. (SATR$$) [E$OLDP]

Operation illegal on directory. PRIMOS

User has tried to perform an operation on a directory that is not allowed (such as editing it). [E$DIRE]

****PA n R-mode function

PAUSE statement _n_ (octal) encountered during program execution.

**** PAUSE n V-mode function

PAUSE statement n (octal) encountered during program execution.

POINTER FAULT 64V mode

Reference has been made to an argument or instruction not in memory. The two usual causes of this are an incomplete load (unsatisfied references), or incomplete argument list in a subroutine or function call.

Pointer mismatch found. PRIMOS

Internal file pointers have become corrupted. No user remedial action possible. System Administrator must correct. [E$PTRM]

PROGRAM HALT AT octal-location R mode and 64V mode

Program control has been lost. The program has probably written over 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 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. (ATCH$$, CREA$$, GPAS$$, PRWF$$, RDEN$$, SATR$$, SRCH$$, SGDR$$). Consult system manager.

The remote line is down.	File System

Remote call-in access to computer not enabled. [E$RLDN]

****RI	R-mode function

Argument is too large for real-to-integer conversion. (C$12)

****RN n	R-mode function

Device error or end-of-file in 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. [SQ] [E$SDER]

Segdir unit is not open.	File System

Attempt has been made to reference a segment directory which 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.

****SQ	R-mode function

Negative argument in SQRT or DSQRT function.

**** SQRT - ARGUMENT<0	V-mode functon

Negative argument in SQRT function.

****ST n	R-mode function

STOP statement n (octal) encountered during program execution.

**** STOP n	V-mode function

STOP statement n (octal) encountered during program execution.

****SZ	R-mode function

Attempt to divide by zero (single-precision).

Stack overflow in crawlout.	PRIMOS

System error. [E$CROV]

Too many subdirectory levels.	File System

Attempt to create more than 72 levels of sub-UFDs. This error occurs only on old file partitions; new file partitions have no limit on UFD levels. [E$TMUL]

UFD FULL	Old file call

No more room in UFD. [SK]

The UFD is full.	File System

UFD has no room for more files and/or sub-UFD's. Occurs only in old file partitions. (CREA$$, SRCH$$) [E$FDFL]

UFD OVERFLOW	Old file call

No more room in UFD.

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 file on PRIMOS file unit already in use. [SI].

Unit in use. — File System

Attempt to open file on 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. [PD, SD]

Unit not open. — File System

Attempt to perform operations with a file unit number on which no file has been opened. (PRWF$$, RDEN$$, SRCH$$, SGDR$$). [E$UNOP]

UNIT OPEN ON DELETE — Old file call

Attempt to delete file without having first closed it. [SD]

****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 WARNINGS AND MESSAGES

Bad $$ command.

(Fatal) A command file was submitted using the JOB command that had a $$ line other than the $$ JOB line as the first non-comment line. The command file should be changed so that the "$$" line is legal. The use of $$ is reserved for future expansion by BATCH.

(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 will execute with the specified changes in place. The job status will be displayed after the above message is typed out.

Command file required as first argument on submission.

(Fatal) The JOB command was given with job options (such as -HOME, -PRIORITY, -CPTIME, etc.) but no command file was seen before those options. The syntax is "JOB <u>pathname</u> [-options]".

Cpu limit must be specified.

(Fatal) The queue referred to by a -QUEUE option during job submission is defined such that the -CPTIME option is a required parameter (i.e., default CPU limit for that queue is <u>greater</u> than the maximum CPU limit for that queue). The job should be resubmitted with the -CPTIME option specified. To determine the maximum limits for queues, use BATGEN -DISPLAY.

Elapsed time limit must be specified.

(Fatal) The queue referred to by a -QUEUE option during job submission has a default elapsed time limit greater than its maximum time limit. Resubmit the job with the -ETIME option specified.

End of line.

(Fatal) One of the Batch programs was expecting to find more information on the command line, but end-of-line was found instead. The message will generally contain more information on what was expected. Re-enter the command with the additional requested information.

End of line. Illegal <option> argument

(Fatal) One of the job parameter options specified on the JOB command line had no argument. The information required by that option should be supplied when the command is re-entered.

File has no non-comment lines. <filename> (JOB)

User has tried to submit a CPL program or command file that contains no commands. (The file either is empty or is made up entirely of comments.)

Home ufd required.

(Fatal) The -HOME option was not present on the JOB or the (optional) $$ JOB line during submission, and the program was unable to determine the home attach point of the submitting job. Resubmit the job, and include the -HOME option followed by the absolute pathname of the UFD where the job is to execute. If the pathname cannot fit, use a shorter version of it when you resubmit the command file, after editing the file to include an "ATTACH" command that descends the remaining sub-ufds to reach the destination.

Home=<pathname>

(Response) During job submission, the -HOME option was not specified on the command line or in the command file ($$ JOB), so the JOB command determined the home attach point of the submitting job. This message is typed out to remind the user that the -HOME option was not specified. The job did successfully submit, however.

Illegal -CHANGE option.

(Fatal) The options -QUEUE and -PRIORITY are illegal during a -CHANGE operation using the JOB command, as queue and queue priority of a job cannot be changed. Cancel or abort the job and resubmit it into the appropriate queue with the desired queue priority.

Illegal combination. <option>

(Fatal) A job parameter (such as -ACCT, -HOME or -QUEUE, etc.) was specified on the same JOB command line as an option to perform an action (such as -CANCEL, -DISPLAY, -ABORT, etc.). Use separate JOB commands to perform separate functions.

Note

This message can also result from giving the -FUNIT option for a CPL program. CPL files cannot specify FUNITS.

Illegal limit.

(Fatal) The parameters supplied to the -CPTIME or -ETIME options during job submission/changing were not legal limits, i.e. they were less than or equal to Ø, or were not legal decimal numbers and not the string "None". Re-enter the command with legal limits.

Illegal name.

(Fatal) One of the Batch programs was expecting a name or command, but it read an unquoted token beginning with a dash ('-'), indicating that an option was present.

Illegal number. <text> (JOB)

(Fatal) The argument for the -FUNIT or -PRIORITY option during job submission using the JOB command was not a legal decimal number. Re-enter the command line with legal numeric parameters.

Illegal option.

(Fatal) One of the Batch programs was expecting an option, i.e., an unquoted token beginning with a dash ('-'). Re-enter the command line with a legal format.

Illegal queue name. <text> (JOB)

(Fatal) The queue name specified after a -QUEUE option while submitting or changing a job did not comply with queue name format rules. Use BATGEN -STATUS or -DISPLAY to determine the names of legal queues.

Incorrect user-name.

(Fatal) A command file was submitted using the JOB command that had a $$ JOB line as the first non-comment line, but the user-name specified after the "JOB" specifier did not match the user-name of the submitting user. Edit the command file and change the user-name in the $$ JOB line to the user-name of the submitter.

*** Invalid batch database, please contact your system administrator.

(Severe) The running job detected an error (such as disk failure, pointer mismatch, or misprotected file) in the Batch system database. It will flag the database as invalid. Notify the System Administrator, who has the responsibility for re-initializing the database (or running *FIXBAT or FIXRAT as the case may be). The BATCH and JOB commands will be inoperative until the situation is resolved.

<nn> is out of range. <option>

(Fatal) The numbers supplied as parameters to the -FUNIT or -PRIORITY options during job submission/changing were out of range. The range for -FUNIT is from 1 to 126; that for -PRIORITY is from 0 to 9. The job should be resubmitted or changed with legal -FUNIT and -PRIORITY values. Note that the system may be configured to have fewer than 126 units per user at cold-start, and the -FUNIT argument will be limited to the maximum configured unit number.

?JOB <extnam>(<intnam>) <status>.

(Warning) An attempt was made to perform an operation on a job using the JOB command that could not be performed because of its status: for example, trying to restart a completed job.

Job name required.

(Fatal) The options -CHANGE, -CANCEL, -ABORT, -RESTART, -HOLD and -RELEASE all require a job identifier (internal or external name). Re-enter the command with the job id. (For example: "JOB C.TOP -HOLD", "JOB #10032 -ABORT").

Job not found.

(Fatal) The job referred to in a JOB command such as -CHANGE, -CANCEL, -ABORT, -RESTART, -HOLD or -RELEASE, could not be found by searching the active jobs list. This could mean one of three things: that no job exists with that name, that all jobs that have that name are not active jobs (i.e., have completed, aborted or been cancelled), or that a job exists with that external name but the user making the request is not the same user that originally submitted the job.

(Job no longer restartable)

(Response) A JOB -CANCEL was performed on an executing job. The job itself is not cancelled; it has been flagged as being unrestartable (i.e., a -RESTART will abort the job but not restart it).

(Job not restartable)

(Warning) A JOB -RESTART was performed on a job that had been flagged as unrestartable. An attempt will be made to abort the job.

(Job restarted)

(Response) A JOB -RESTART was performed on a job, and the job has been flagged as restartable. Although an error message may appear after this message, the job will generally be restarted unless a JOB -CANCEL or JOB -CHANGE -RESTART NO is done on it. Possible errors after this message include "Insufficient access rights" if the user is logged in as SYSTEM, and restarted another user's job from a user terminal (not the supervisor terminal), or if the process recently logged out. "Not found" may also be returned in this case.

*** Jobs are not being processed at this time.

(Severe) If followed by "*** Please contact your system administrator immediately", it indicates that the Batch database has not been initialized, or that something has happened to it (like a disk head crash). If followed by "*** Please try again later", it indicates that while the database is still valid, the Batch monitor was logged out using a method other than "BATCH SYSTEM -STOP", and will verify the validity of the database when it is started up. Either way, the user will be immediately returned to command mode (i.e., the operation the user attempted will <u>not</u> be performed). This can be typed out by the BATCH or the JOB commands when they start running.

Multiple jobs with this name (use internal name).

(Fatal) A reference was made to a job using a filename in the JOB command, and there were at least 2 such jobs belonging to the user making the reference that were active. The job-id must be used in this case. Use JOB -STATUS ALL to determine the filenames and job-ids of all jobs belonging to the user issuing the command.

Multiple occurance.

(Fatal) An option was specified twice during job submission or job changing (example: JOB C_TEST -HOME HERE -HOME THERE) on either the JOB or $$ JOB line. (If the option is specified once on the JOB line and once on the $$ JOB line, no error will result and the parameter on the JOB line will take precedence). Re-enter the command, specifying each option only once.

Must be first option.

(Fatal) The options -CHANGE, -CANCEL, -ABORT, -RESTART, -STATUS, -DISPLAY, -HOLD and -RELEASE must be the first option on the JOB command line (after a sometimes optional job identifier). Use the JOB command several times to perform several operations.

No active jobs [named "<jobname>"] for user <username>.

(Response) There are no jobs belonging to that user that are waiting, held, or executing.

The jobname is output if a jobname was specified for the -DISPLAY or -STATUS command; otherwise it is omitted.

No job changes specified.

(Fatal) The -CHANGE option was given to the JOB command, but no actual changes were specified on the command line. Specify changes to be made after the -CHANGE option.

No jobs [named "<jobname>"] for user <username>.

(Response) This message is typed out by a JOB -DISPLAY ALL or -STATUS ALL command, and indicates that there are no jobs belonging to that user.

No longer executing.

(Fatal) A JOB -ABORT or JOB -RESTART was 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, it had disappeared. If the message "(Job restarted)" had been typed out, then the job would be restarted.

No queue available for job.

(Fatal) A job was submitted using the JOB command that did not use the -QUEUE option to specify the queue to which it was to be submitted, and no suitable queue could be found. Suitability for a queue includes CPU and elapsed time limits being within the confines of the queue, queue being unblocked, etc. Use the BATGEN -STATUS or -DISPLAY command to yield a list of legal queues and their status.

No queues have waiting or held jobs.

(Response) A BATCH -DISPLAY command was issued, and there were no queues that had any waiting or held jobs in them. A queue may have one executing job in it, but an executing job is not considered a waiting or held job.

No recent jobs [named <jobname>"] for user <username>.

(Response) There are no jobs belonging to that user (or in the batch system if the user is SYSTEM) that were submitted, initiated, aborted, completed or cancelled today.

No running jobs.

(Response) A BATCH -DISPLAY command was issued, and there were no jobs that were currently running. It is possible for there to be no running jobs and to have jobs waiting, however, even when the monitor is running and there are free phantoms; there is always a small amount of turnaround time between the submittal of a job and the execution of a job.

Not an absolute treename.

(Fatal) The home ufd specified with the -HOME option during submission using the JOB command (or changing of job parameters) was a relative (pathname), i.e., it began with "*>". Re-submit the job, giving an absolute pathname after the -HOME option.

Not your job.

(Fatal) A reference was made to a job using an internal name in the JOB command, and the referenced job did not belong to the user making the reference. Use "JOB -STATUS ALL" to obtain a list of all jobs belonging to the user making the request.

Null home ufd.

(Fatal) The home ufd specified with the -HOME option during submission using the JOB command (or changing of job parameters) was a null string. Re-submit the job with an absolute pathname after the -HOME option.

Please stand by.

(Response) This message and others like it ("File in use, please stand by") will be output if the program being run is trying to gain access to a file that is in use for more than 5 seconds. After 20 seconds, the "File is use..." message will be output, and after 30 seconds, the message "Timeout of 30 seconds has occurred" will be output and the program will "give up". Usually this will result in a fatal error, as it could indicate that system security is broken.

Please wait.

(Response) This message asks that the user be patient because the program he is running has been locking up the Batch database too long and is not allowing other processes to have access to it. It is not a fatal error. It generally only is output when a system is heavily loaded, or when the current process has a very low priority and does not run frequently.

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.

Queue deleted.

(Fatal) The queue that the job was being submitted to was present when it was first checked out, but by the time the command file had been 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 or -DISPLAY command should provide a list of currently available queues and their status, if the file that defines queues is accessible by users.

Queue full.

(Fatal) There are already 10,000 jobs (whether active or inactive) in the queue to which the job is being submitted. The queue must be deleted and re-created before more jobs can be submitted to it. The system administrator should be asked to do this. Meanwhile, if any other queues are available, they can be used instead by the user.

Register setting.

(Fatal) Register settings are illegal in the Batch subsystem (except as part of a submitted command file). Re-enter the command line without the register setting.

Searching for free command file, please stand by.

(Response) This and other messages like "Queue is in heavy use...please stand by" mean that many users are submitting command files at once. The situation should resolve itself in a short amount of time.

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 will be preceded by a message indicating the maximum limit for that queue ("Cpu limit is xx" or "Elapsed time limit is xx"). If the limits cannot be lowered and the job successfully run, then try a queue with higher limits.

Syntax error. Register settings are illegal

(Warning) This message is output if end-of-line is expected and a register setting is found instead. Re-enter the command without register settings.

<text> seen when end-of-line expected.

(Fatal) <text> was seen when there should have been no more text (end of line). The command will be ignored and the user will be returned to PRIMOS level.

This job cannot be restarted.

(Response) Output by a JOB -DISPLAY command if the job being displayed has had a JOB -CANCEL done to it while it was executing, or was submitted with the -RESTART NO option. Any -RESTARTs done to the job will abort the job (if they succeed), but the job will not be restarted.

(This job has already executed nn time(s)).

(Response) Output by a JOB -DISPLAY command if the job being displayed is executing and has already been executed. This is the result of a JOB -RESTART being done on that job, or a system cold-start after being brought down while the job was executing.

Too many options.

(Fatal) At least two options were entered that conflicted with each other, such as JOB -DISPLAY -CHANGE or JOB C_TEST -ABORT -CANCEL. Use separate JOB commands to perform separate operations.

Unknown option.

(Fatal) An option was entered to the BATCH or JOB command that was not recognized.

Warning: the Batch monitor is still awaiting start-up instructions from the operator, so jobs are not yet being processed.

The Batch monitor's phantom is running, but the operator has not yet started the Batch subsystem. When the operator gives the start command, the job will be submitted for execution.

Warning: jobs are not being processed at this time.

(Response) The Batch monitor is not running. No submitted jobs will be executed until it has been started up. The operation requested will then be performed. If the monitor is force-logged out, or the system is shut down without the monitor logging itself out, there may be a database problem as a result.

APPENDIX E

EDITOR COMMAND SUMMARY

The following is an alphabetic list of each Editor command and its function. Acceptable command abbreviations are underlined. 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 beyond the last line of the file.
BRIEF	Speeds editing by suppressing the (default) verification responses to certain Editor 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 line containing string.
DUNLOAD filename [n]	Deletes n lines from 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, EDITOR writes into the current file and prints its name.
FIND string	Moves the pointer down to the first line beginning with string.
FIND(n) string	Moves the pointer down to first line with string beginning in column n.
GMODIFY	Allows the user to enter a string of subcommands which modify characters within a line.
IB string	The "INSERT BEFORE" command inserts string as a new line immediately before the current line.
INPUT (ASR) (PTR) (TTY)	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 current line.
KILL character	Sets kill character to character.
LINESZ [n]	Changes maximum line length. (Minimum linesz is 1Ø). Linesz changes the maximum length of both command lines and input lines.
LOAD filename	Loads filename into 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 end of file.
MODE CKPAR	Prints characters as real characters if parity's on, as octal numbers (^nnn) if parity's off.

MODE COLUMN	Displays column numbers whenever INPUT mode is entered.
MODE COUNT start increment width	PRINT BLANK SUPPRESS Turns on the automatic incremented counter.
MODE NCKPAR	Prints 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 printed line.
MODE NNUMBER	Turns off the line number display (default).
MODE PRALL	Prints lower case characters if device has that capability.
MODE PRUPPER	Prints all characters as upper case. Precedes lower case characters with an ^L and precedes upper case characters with an ^U if the device is upper case only.
MODE PROMPT	Prints prompt characters for INPUT and EDIT modes.
MODE NPROMPT	Stops printing of INPUT and EDIT prompt characters (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 buffer-2 /string/	Move 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 pointer down to first line NOT beginning with string.
NFIND(n) string	Moves pointer down to 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 current line. Use tabs to start in middle of line. use ! to delete existing characters. (A blank in the string leaves the old character in place.)
PAUSE	Returns to operating system without changing the Editor state.
POINT line-number	Relocates the pointer to line-number.
PP [first] [last]	The POSITION PRINT command prints a range of lines relative to the current position without changing the current position: first — number of lines away from current position to start printing last — relative number of lines away from the current position to stop printing 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 position (first) and the default ending line is the current line.

	If no numbers are given the default PP -5 5, which prints from five lines above to five lines below the current position.
PRINT [n]	Prints the current line or n lines beginning with the current line. Moves pointer to last line printed.
PSYMBOL	Prints 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) (PTP) [n]	Punches n lines on high- or low-speed paper-tape punch.
QUIT	Returns control to PRIMOS without filing text. If file has been modified EDITOR warns user and asks "OK TO QUIT?"
QF	The "QUIT FINAL" command lets the user QUIT out of a modified file without having the EDITOR ask if it may throw away the work file.
RETYPE string	The current line is replaced by string.
SAVE [filename]	Saves file without leaving EDITOR. If user does not specifiy filename, EDITOR saves into the file being edited and prints its name.

SYMBOL name character	Changes a symbol name to character. Current default values are:
TABSET tab-1...tab-8	Sets up to eight logical tab stops to be invoked by the tab symbol ( ).
TOP	Moves the pointer one 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	Prints the current line number.
XEQ [buffer]	Executes the contents of buffer. If no buffer name is given, the last command line is re-executed.
*[n]	Causes 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 file is reached.

Default values for SYMBOL:

Name	Default Characters
KILL	?
ERASE	"
WILD	!
BLANK	#
TAB	\
ESCAPE	^
SEMICO	;
CPROMPT	$
DPROMPT	&

Index

INDEX