MAN2604

PRIMOS FILE SYSTEM
User Guide

Revision A
January 1977

PRIME
Computer, Inc.
145 Pennsylvania Ave.
Framingham, Mass. 01701

Copyright 1976 by

Prime Computer, Incorporated

145 Pennsylvania Avenue

Framingham, Massachusetts Ø17Ø1

Performance characteristics are
subject to change without notice.

CONTENTS

	PAGE
SECTION 1 FILE SYSTEM OVERVIEW	
CONCEPTS	1-1
USING THE FILE SYSTEM	1-2
SECTION 2 FILE STRUCTURES	
FILE SYSTEM	2-1
TYPES OF FILES	2-1
DIRECTORIES	2-5
FILES AND DISK STRUCTURE	2-6
FILE HANDLING SUBROUTINES	2-8
FILE ACCESS	2-10
OPERATING SYSTEM USER INTERACTION	2-12
SECTION 3 FILE SYSTEM SUBROUTINES	
INTRODUCTION	3-1
CALLING LOADING LIBRARY SUBROUTINES	3-1
CALLING SEQUENCE NOTATION	3-1
FILE SYSTEM TERMINAL I/O SUBROUTINES	3-2
ATTACH	3-4
BREAK$	3-7
CMREAD	3-8
CNAME$	3-9
COMINP	3-10
COMANL	3-11
COMINP	3-12
COMANL	3-13
COMEQV	3-13
D$INIT	3-14
ERRSET	3-15
EXIT	3-16
FORCEW	3-16
GETERR	3-16
GETWRD	3-17
GINFO	3-17
NAMEQV	3-18
PRERR	3-18
PRWFIL	3-19
PUTC	3-24
RDCOM	3-24
RECYCL	3-24
RESTOR	3-25
RESUME	3-25

RREC 3-26
SAVE 3-29
SEARCH 3-30
T1IN 3-36
TIOU 3-36
TNOU 3-36
TNOUA 3-37
TOOCT 3-37
TIMDAT 3-38
T$CMPC 3-39
T$LMPC 3-41
T$MT 3-43
T$SLC 3-44
T$VG 3-48
NUMBER INPUT OUTPUT 3-55
UPDATE 3-57
WREC 3-58

SECTION 4 FILE UTILITY COMMAND (FUTIL)

FILE STRUCTURE (TREENAMES) 4-1
DESCRIPTION OF FUTIL COMMANDS 4-3
QUIT 4-4
FROM 4-5
FROM* 4-6
TO 4-7
ATTACH 4-7
COPY 4-8
COPYSAM 4-9
COPYDAM 4-9
TRECPY 4-10
UFDCPY 4-11
DELETE 4-12
TREDEL 4-13
UFDDEL 4-13
LISTF 4-14
FUTIL RESTRICTIONS 4-17

APPENDIX A FILE AND HEADER FORMATS

FILE FORMAT A-1
UFD FORMAT A-2
DSKRAT FORMAT A-4

APPENDIX B APPLICATION EXAMPLES

GENERAL INFORMATION B-1
FILE SYSTEM PERFORMANCE B-4
DISK ACCESS TIME B-5
FILE SECURITY B-5

APPENDIX C USE OF PRIMOS FILE SYSTEM

ILLUSTRATIONS

Figure No.	Title	Page
2-1	Hypothetical PRIMOS File Hierarchy with SAM and DAM File Structures	2-4
2-2	Memory Allocation in 16K System	2-18
4-1	Sample File Structure	4-2
4-2	Typical Traverse of Directory Tree by FUTIL	4-16
A-1	UFD File Format and Use	A-3
D-1	Control Characters	D-3
D-2	Notes on Internal ASC11	D-7

TABLES

Table No.	Title	Page
2-1	Memory Areas and PRIMOS II File Units	2-17
3-1	Structure of SMLC Hardware Configuring Block	3-47
3-2	Maximum Buffer Length for Printer/Plotters	3-52
A-1	File and Header Formats	A-1
A-2	UFD Formats	A-2
A-3	Format of DSKRAT	A-4
D-1	ASC11 Communications Codes	D-2
D-2	Internal ASC11 Codes	D-3

DOCUMENT DESCRIPTION

This document describes the fundamental features of the file system associated with the PRIMOS II, III, and IV operating systems.

Section 1	Is an overview of PRIMOS file system concepts and usage.
Section 2	Describes PRIMOS file and directory file structures, file system usage, disk and file system relationships, and memory usage.
Section 3	Contains detailed descriptions of subroutine calls related to file system operations and terminal I/O. This section emphasizes the calls that are executed intrinsically by the operating system (internal).
Section 4	Is a discussion of the File Utility (FUTIL) and its file manipulation commands and messages. This section also contains a definition and examples of file treenames.
Appendix A	Describes the format of PRIMOS file record headers and the physical organization of PRIMOS files; it also describes UFD and DSKRAT formats.
Appendix B	Discusses application examples that apply features of the PRIMOS file systems to handle complex data structures.
Appendix C	Provides examples that show how to create segment directories and DAM files.
Appendix D	Is a definitive example of using the subroutines SEARCH, PRWFIL, and ATTACH.

RELATED PUBLICATIONS

The following Prime documents should be available for reference:

Title	Manual No.
Prime CPU System Reference Manual (instruction set, addressing modes, input/output programming)	MAN1671
Macro Assembler Language Reference User Guide	MAN1673
FORTRAN IV Reference Guide	MAN1674
Program Development Software User Guide (Editor, Loader, Debugger, etc.)	MAN1879
Library Subroutine User Guide	MAN1880
PRIMOS Interactive User Guide	MAN2602
PRIMOS Computer Room User Guide	MAN2603
PRIMOS IV Systems Reference Manual	MAN2798

Filename Conventions:

B←XXXX	Binary (Object) file
L←XXXX	LISTING file
C←XXXX	Command file
XXXXXX	Source file
*XXXXX	SAVED (Executable) file (PRIMOS LOAD)
#XXXXX	SAVED (Executable) file (PRIMOS IV SEG)

NOTE:

On some printing devices, the back-arrow character (←) is printed as an underline (_).

SECTION 1

FILE SYSTEM OVERVIEW

CONCEPTS

The following paragraphs define terms used in describing a disk-based operating system (PRIMOS II, PRIMOS III, or PRIMOS IV). The generic term PRIMOS refers to all three systems.

File

A file is a named set of information organized and stored on a magnetic disk in such a way that a computer program can use the information.

For PRIMOS, a file consists of a list of 16-bit binary words; a binary word is the smallest item of information that can be moved to or from a file at one time.

File Access

The process of moving information from a file stored on disk to a location in high-speed memory is called reading from a file. Moving information from a location in high-speed memory to a file stored on disk is called writing to a file.

File Creation

Files may be made through the use of a system editor at a terminal (Teletype or CRT keyboard type); they may be made by copying information stored on paper tape, punched cards, etc., or they may be generated by computer programs.

Some Typical File Content

1. Lists of employee names, addresses, salaries, etc. stored as files for payroll and bookkeeping programs to use.

2. Computer programs coded in languages that may be read by humans and stored as files so that other programs may be used to translate the human-readable program into a program that is both meaningful to a computer and can be run on a computer.

Purpose of File System

The purpose of having a file system is to simplify the manipulation of large quantities of data using the computer. The major goals of the file system are listed below:

1. File creation without manual pre-allocation of the storage medium.

2. Ability to reference a file by name.

3. Clustering like files together.

The first goal is implemented by keeping a file on each disk that lists the available space for the disk. Since the whole process is automatic, the average user need not concern himself with this process, other than to know that it works.

Referencing files by name means the desired file may be selected for operation by giving the system an array of alphanumeric characters. The file system does this by having a file that is used as a directory; it contains the names of other files and their locations on the disk. The system can find this Master File Directory (MFD) readily because both its name and its location are always the same.

The third goal is achieved in two ways. The first is to have many file directories; this allows like files to have their names and locations saved in the file directory. The second way is by nesting file directories; i.e., some of the filenames saved in a file directory can be the names of other file directories. Thus, files may be classified to an arbitrary extent.

A side effect of clustering files in a file directory (files that have their names stored in a file directory are often said to be ´in´ the directory) is that some degree of access protection can be built in by associating a password with each file directory. To examine the files in a directory, the user must first supply the password for that directory.

<u>Summary</u>

A file directory is a file that contains the names of other files on the disk and the location on the disk of these files. A file directory may contain the names of other file directories. To access files stored in a directory, the user must give the password for that directory.

USING THE FILE SYSTEM

To access files, the user must be attached to some file directory. A user is properly attached when the file system has been supplied with the proper file directory name and password, and it has found and saved the name and location of the file directory. It can therefore find and operate on all files contained in that file directory.

The major operations on files are: initialization for access (open); access; shutdown and resource deallocations (close); and deletion.

Opening a File

A file may be opened for reading only, for writing only, or for both reading and writing. If a file is opened for reading only, it may be read, but it cannot be changed.

The operation of opening a file does the following :

1. Searches the file directory to see if the filename requested is there;

2. Sets up tables and initializes buffers in the operating system; and

3. Defines a pseudonym for the file. This pseudonym is called the file unit number, and is the only name used for transfer of data to and from the file.

If a file is opened for writing only, or for reading and writing, it may be changed; if the filename is not found in the directory, the filename is added to the file directory, and a new file is created. When a new file is created at the time of opening, no information is contained in the file.

Using an Open File

Once a file has been opened, a file pointer is associated with the file. The file pointer indicates the next binary word to be accessed. To understand how the file pointer works, imagine that the words in a file are serially numbered from Ø. The file pointer is then the number of the next word to be accessed in a file.

Access

On an open file, information may be read from the file starting at the file pointer into high-speed memory, or information may be written to the file starting at the file pointer.

Access and File Pointer

When a file is accessed, the file pointer is incremented once for each binary word accessed.

Position

The file pointer may also be moved backward and forward within a file without moving any data. This is called positioning a file. The value of a file pointer is called the position of the file. Positioning a file to its beginning is often called rewinding a file.

Truncation

It is possible to shorten a file by truncating it. When a file is truncated, the part of the file that is at or beyond the file pointer is eliminated from the file. If the file pointer is positioned at the beginning of the file, all of the information in the file is removed but the filename remains in the file directory.

Closing a File

A file that has been opened may be closed. The file unit number (pseudonym) and the corresponding table areas in the operating system are `cleaned up´ and released for reuse.

Deleting a File

A deleted file has its filename removed from the file directory, and all of the disk memory that the file occupied is released for use by other files.

Physical Disk Consideration

A disk storage medium is composed of many separate blocks of data recording space (disk records or sectors). How these blocks are put together to make a file can affect the efficiency of positioning by several orders of magnitude. Because of this, the file system has two different ways of linking physical disk records together to form a file. One way, SAM (Sequential Access Method), results in more compact storage on the disk and requires less high-speed memory for efficient operation, but is much slower for repeated random positioning over a file. The other way, DAM (Direct Access Method), results in quicker positioning over a file, but requires more disk space and more high-speed memory. SAM and DAM files are functionally equivalent in all other respects.

More on File Directories

File directories were previously described as files containing the names and locations of other files on the disk. This kind of file directory is referred to elsewhere in the documentation as a User File Directory (UFD). The file system supports a second kind of file directory called a segment directory. Segment directories differ from UFD´s in one fundamental respect: they contain file locations but not file names. As far as the file system is concerned, the files in a segment directory have no names. This means that the file system user is charged with all of the bookkeeping involved in the use of a segment directory.

Segment Directory Use

Each binary word in a segment directory is assumed to hold a legitimate file location on the disk. The segment directory file is opened for reading/writing on a unit of the user´s choice. The segment directory

file is then positioned to the word containing the desired file location.

A desired file may be opened, closed, deleted, or truncated by giving the file unit number of the segment directory file rather than the filename.

SECTION 2

FILE STRUCTURES

FILE SYSTEM

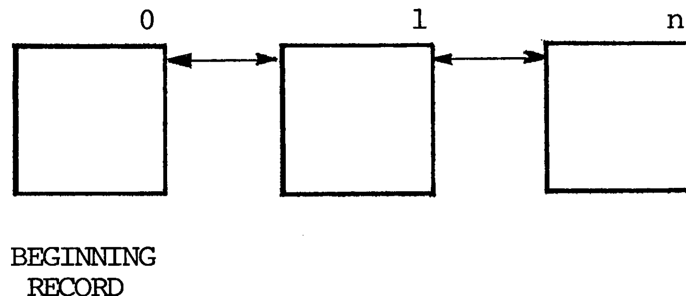
The PRIMOS II, III, and IV file systems consist of a hierarchy of files and file directories. There are two types of files and two types of directories. These are described in this section.

TYPES OF FILES

The two types of files are: Sequential Access Method files (SAM files) and Direct Access Method files (DAM files). The structural differences between these two file types are transparent to the user.

SAM Files

A SAM file is the basic way of structuring disk records into an ordered set; i.e., a threaded list of physical disk records. The following example shows this structure:



SAM File Structure

In PRIMOS II, a SAM file consists of a collection of disk records chained together by forward and backward pointers to and from each record (See Appendix A). Further, each record in a SAM file (or any file) contains a pointer to the beginning record address (BRA) of the file. The file system maintains the record headers and is responsible for the structure of the records on the disk.

Figure 2-1 shows an example of how SAM files may be related within a PRIMOS II file hierarchy.

DAM Files

A DAM file is a direct access file. DAM file organization uses the SAM file method of making an ordered set; for purposes of rapidly accessing the i'th data record, a special technique is used:

1. Physical disk record Ø of a DAM file is reserved for use by the system. No user data is ever written in this record.

2. The first disk record (logical record Ø) to contain user-written data is the second record of the threaded list of disk records. The first disk record (D) contains pointers to the second, third, ... 44Øth disk record of the file, as shown in the following example:

DAM File Structure

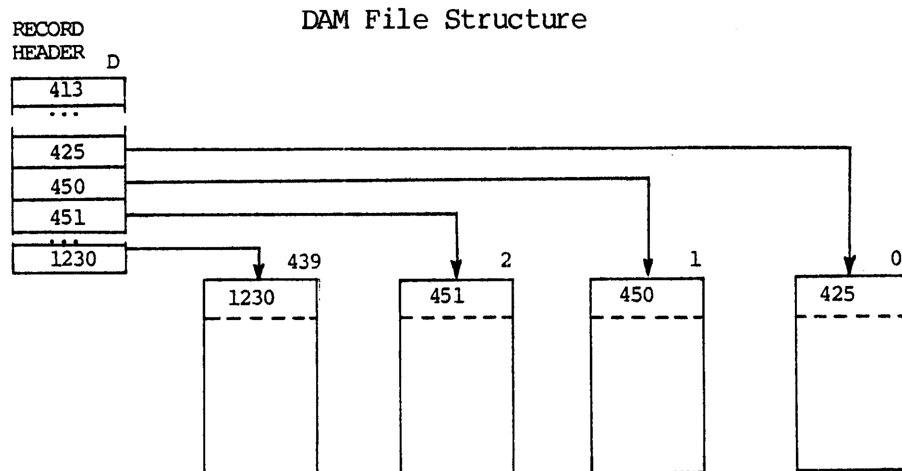


Figure 2-1 shows a typical relationship of DAM files within a PRIMOS II file hierarchy.

A DAM file can continue to grow beyond 44Ø records. For PRIMOS, the records beyond the 44Øth will be threaded and referenced as if they were records in a SAM file. For example, to access the 445th record of a DAM file, the file system would go to the 44Øth record directly and seek through the remaining five records sequentially. For an example of how to create a DAM file, refer to Section 4.

File Records

All files on PRIMOS disks are stored in fixed-length 448-word records, (1Ø24-word records for storage module disks), chained together by forward and backward pointers.

In PRIMOS, the first eight words of a record is the record header (first 16 words for storage module record). Specific content of record

headers is discussed in Appendix A. After the record header, all remaining words within the record may be used to store ASCII character pairs or 16-bit words. For further information about disks and storage modules, refer to the Computer Room User Guide (MAN2603).

File Contents

A file is a series of records of the type described above, with the distinction that the first record in such a chain is reached from a pointer within a User File Directory or an entry in a segment directory.

Every file contains a series of 16-bit words. The format depends on the type of data in the file and how they were originally entered into the file system. The following types of files are in general use in PRIMOS II Systems:

Line Image	ASCII character text, packed two characters per word, as entered from a terminal or from the Prime card reader, paper-tape readers, etc.
Line Image Compressed	Same as above, but successive spaces are replaced by a relative horizontal tab character followed by a space count, and lines are terminated by a LINE FEED character.
Object Format (Relocatable Binary)	Block-format object code as generated by the Macro Assembler and FORTRAN Compiler for processing by linking loader.
Saved Memory Image	Header block followed by a direct transcription of high-speed memory between limits Starting Address (SA) and Ending Address (EA).
Directories (UFD and Segment)	See Appendix A for format details.

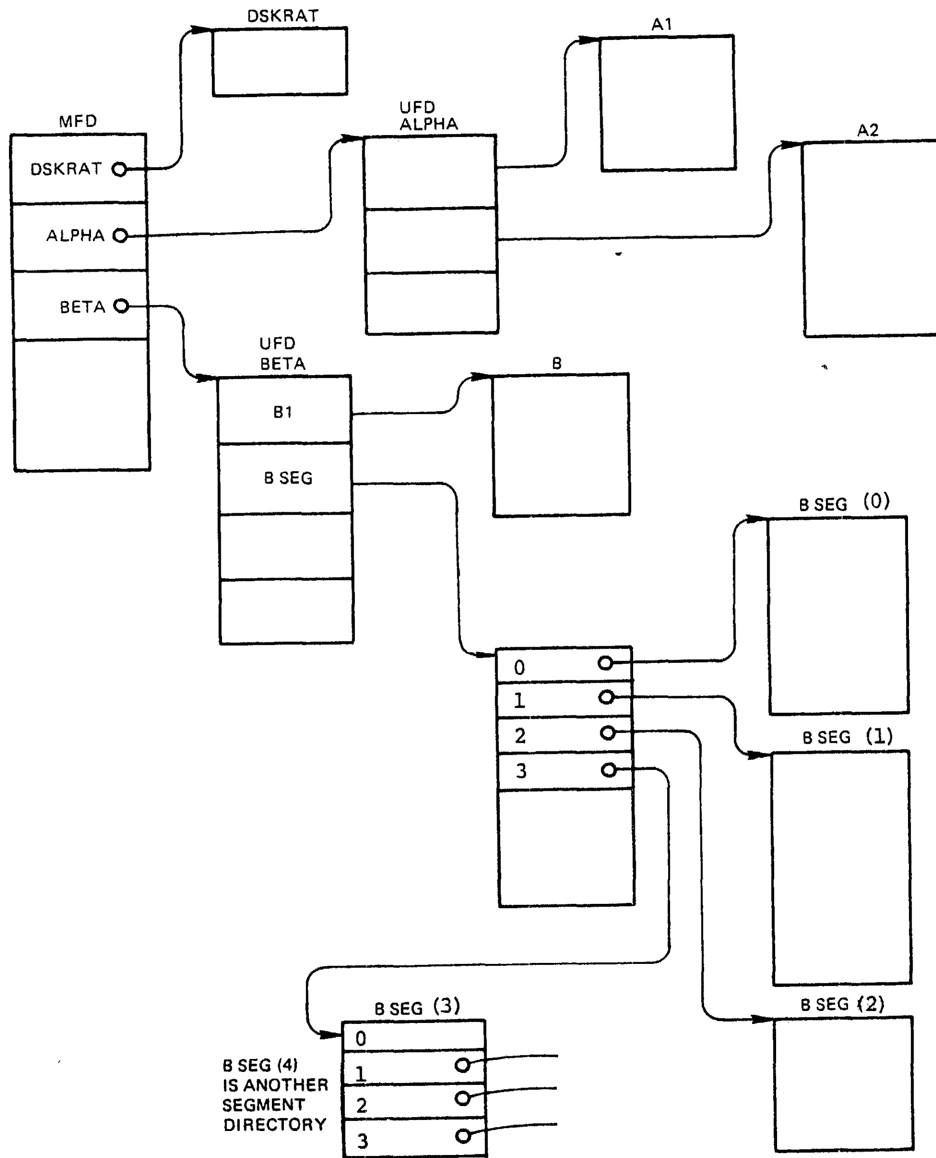


Figure 2-1. Hypothetical PRIMOS File Hierarchy with SAM and DAM File Structures

DIRECTORIES

Directories are specialized files that contain entries that point to files or, in some cases, other directories. Directories are the nodes in the file system tree structure hierarchy, whereas files are the branches. Figure 2-1 illustrates this concept. The types of directory are UFD´s and segment directories.

Segment directories may be organized as SAM files or DAM files, depending on the kind of file structure the user wishes to build.

MFD and UFD

Each disk pack (or device, in the case of non-removable media) has one User File Directory called a Master File Directory (MFD) that contains an entry for each User File Directory (UFD) in the MFD. In turn, each UFD contains an entry for every file or directory file in that directory. UFD´s and MFD´s are accessed in the same way as other files.

Master File Directory (MFD)

Each disk unit contains one MFD file as an index to the first physical record of each UFD in the MFD. The MFD has the same format as any UFD. The first record of the MFD begins at physical record one of the disk. Figure 2-1 shows a chain of pointers extending from the MFD to UFD and segment directories, and to a DAM or SAM file.

User File Directory (UFD)

A User File Directory (UFD) is a file that links PRIMOS filenames to the physical record of a file.

A UFD, in the format shown in Appendix A, is associated with each system user. The UFD header consists of a word count followed by the password. After the header, the UFD contains an entry for every file or directory named by the user. Each entry consists of a filename and a word that contains the address of the first physical record of the file (called the beginning record address or BRA). Currently, each UFD can contain up to 72 entries (179 entries in a storage module). The first physical record of each UFD is accesssed from a pointer (entry) in a UFD. Details of the contents of the UFD header and entries are given in Appendix A.

Segment Directory

A segment directory is formatted just like a UFD except that instead of several words per entry, there is only one: the pointer to the beginning record of a file. For information on how to create a segment directory, refer to Appendix C.

FILES AND DISK STRUCTURE

Disk Record Availability Table (DSKRAT)

PRIMOS maintains a file, formerly named DSKRAT, that stores the status of every physical record on the disk. The name of this file may also be the name of the disk (which is referred to as the Packname). For example, the name of the documentation disk is DOCUME, and the name of the DSKRAT file for this disk is DOCUME. Each record is represented by a single binary bit; a `1´ means the record is available, and a `0´ means it is in use. On a typical PRIMOS disk, the DSKRAT file is allocated several records. The DSKRAT file is maintained as a file on the disk, starting at physical record 2. The format of DSKRAT is shown in Appendix A.

Disk Organization

PRIMOS supports the use of all the Prime disk options. Prime software provides facilities for keyed indexed direct access files (KIDA) (refer to the Subroutine Library Manual (MAN 1880) for details).

Multiple disks are organized so that every fixed disk and every removable disk is a self-consistent volume with its own bootstrap, DSKRAT, and MFD.

Logical record zero is cylinder zero, head zero, sector zero on all options except the dual (fixed and removable) device, which has two logical zeros -- one on a fixed disk (head 2) and one on the removable disk (head 0).

File Units

When a disk file is made active for reading or writing, to hold one or two disk records at a time, it must be assigned one buffer in high-speed memory if the file is a SAM file, or two buffers if the file is a DAM file. The buffer, plus associated pointers and status indicators, serves as an access port for the exchange of data between the disk file and the active program. A user generally is concerned with file units; he is not aware of a buffer, except when PRIMOS II runs out of memory or overwrites a user program. One file at a time can be assigned to each unit; therefore, a maximum of 16 files can be active (open) at any one time. The files may be open on several different logical disk units at once. Under PRIMOS III and IV, no space is used in the user´s address space for file buffers.

Opening, Closing File Units

Refer to the discussion on file units, buffers, and opening files in Section 1.

Various ways are provided to associate a specific filename (Filename) to one of the memory buffers (Funit numbers). One method is the OPEN command.

Example:

OPEN Filename Funit Status

Filename is the name of a file listed in the UFD to which the user is currently attached; Funit is a PRIMOS II file unit number ('1-'17), and Status is 1 for reading, 2 for writing, etc.

NOTE:

> The character ' is used to denote an octal number. (For full information, see the Interactive User Guide "OPEN", or Section 3, "SEARCH").

In response to this command, PRIMOS II selects an unassigned buffer area, assigns one or two buffers the specified Funit number, and uses it as the data buffer when reading from or writing to the named file. Whether one buffer or two buffers are assigned depends on whether Filename specifies a SAM file or a DAM file. The file is then said to be open. The 448-word or 1040-word memory buffers are allocated downward, starting from the beginning of PRIMOS II itself. PRIMOS II associates a Funit number to the highest unassigned block when a file is opened. From the terminal, the user can open files with the OPEN, BINARY, INPUT, and LISTING commands, and can close them with the CLOSE command. The command INPUT opens Unit 1 for reading (for example, to provide a source input file to the Assembler or Compiler). The BINARY command opens Unit 3 for writing (of the object output) and LISTING opens Unit 2 for writing (of the listing file). The OPEN command allows a user to assign a file to a unit of his choice and specify the activity - reading, writing, or both. For complete descriptions of commands, refer to the Terminal User's Guide. File Units 1 to 15 may be specified by the user.

Unit 16 can be opened by the user only under PRIMOS III and IV; it is used by PRIMOS II for reading and writing of system files such as DSKRAT and User File Directories.

When the user is communicating with the file structure through one of the standard Prime translator or utility programs, files are referred to by name only. PRIMOS, or the program itself, handles the details of opening or closing files and assigning file units. For example, the user can enter an external command such as ED FILE1, which loads and starts the text editor and takes care of the details of assigning the file FILE1 to an available unit for reading or writing.

Because open files are subject to alteration (deliberate or accidental), the user must keep files closed except when they are being accessed. The CLOSE ALL command returns all open file units to a closed and initialized state.

FILE-HANDLING SUBROUTINES

All file handling is done by a collection of special subroutines, some internal to PRIMOS, and others available as library routines. These routines are used in common by PRIMOS and all Prime system software for simplified and uniform file handling. In addition, they can be called from FORTRAN or assembly-language user programs. Some principal routines for PRIMOS are:

ATTACH	Attaches user to a specified UFD or device.
GETERR	Moves n words from the system error vector ERRVEC into a specified array.
GINFO	Moves n words of information about PRIMOS into a special array.
PRWFIL	Reads 16-bit words from a specified file unit to high-speed memory and writes 16-bit words from memory to a specified file unit. (For details, see Section 3.)
RESTOR	Restores to memory an executable program previously filed by a SAVE operation.
RESUME	Restores to memory and starts an executable program previously filed by a SAVE command.
SAVE	Saves a section of high-speed memory as a named file. High and low address limits, the start-execution address, and other key parameters are saved with the program.
SEARCH	Assigns a named file to a file unit and opens the file for reading and writing.

PRIMOS file handling subroutines are described in Section 3.

The ATTACH, RESTORE, RESUME, and SAVE routines have exactly the same functions as the commands of the same name. These and other file and character handling subroutines are described in detail in the Interactive Terminal User Guide.

All of the file handling subroutines called by the user are loaded with the user's program when the FORTRAN library is loaded. Most of these subroutines are interlude subroutines which issue supervisor calls to PRIMOS. The appropriate subroutine in PRIMOS then executes the appropriate file operation.

File Handling in User Programs

The subroutines (refer to Section 3) simplify communication between the PRIMOS file structure and user programs. In FORTRAN programs, for example, the symbolic device unit numbers in formatted READ and WRITE statements can be associated with PRIMOS file units. The following default assignments are set up by the compiler:

FORTRAN Unit (u)	File Unit (Funit)
5	1
6	2
7	3
8	4
9	5
10	6
11	7
12	8
13	9
14	10
15	11
16	12
17	13
18	14
19	15
20	16

Example: to write a record to file Unit 1 (FORTRAN Unit 5), the user could enter the command OPEN filename 1 2. The OPEN command associates the file Filename with the file unit 1 and opens the file for writing (code 2). During subsequent execution of a program containing a formatted WRITE statement such as:

```
WRITE (5, 10) LINE
```

the contents of array LINE are written as one record to the FORTRAN Unit 5 (File Unit 1), according to FORMAT statement 10.

At the program level, a Filename and Funit number can be associated by the PRIMOS subroutine SEARCH (see below):

```
CALL SEARCH (2, 6HTEXTbb, 1, $50)
```

to open the file named TEXT on Funit 1 for writing. Besides maintaining the file directories, SEARCH also initializes the PRIMOS data base when a file is opened and updates it when the file is closed.

Users normally call the IOCS subroutine CONTRL to open or close a file read or written by FORTRAN READ or WRITE statements (refer to the Subroutine Library Manual (MAN 1880)). The appropriate call that replaces the call to SEARCH is:

CALL CONTRL (2, 6HTEXTbb, 5, $50)

FILE ACCESS

Attaching to a UFD

To access files or use PRIMOS utility functions, the user must be attached to a UFD. Typically, during program development, each user attaches to a UFD reserved for program files with the ATTACH command. For further information, refer to The Interactive Users' Guide. Within executable programs, the user can attach to other UFD's; for example, to access data or to call subroutines. At the program level, this is a accomplished by the subroutine ATTACH, described previously. For further information on the ATTACH subroutine, refer to Section 3.

File Access Control

PRIMOS III or IV gives a user (owner) the ability to open file directories to other users with restricted rights to the owner's files. Specifically, the "owner" of a file directory can declare, on a per-file-basis, the access rights a "nonowner" has over each of the owner's files. These rights are separated into three categories:

- Read Access (includes Execute Access)
- Write Access (includes over-write and Append)
- Delete/Truncate rights

The owner of a UFD can establish two passwords for access to any file in the UFD; the owner password and the nonowner password. The owner password is required to obtain owner privileges. The nonowner password (if any) is required to obtain nonowner privileges.

The command:

PASSWD Owner-Password Nonowner-Password

replaces the existing passwords in the UFD with a new owner-password and a nonowner-password. This command must be given by the owner while attached to the UFD. A nonowner cannot give this command. The command:

PROTECT Filename Okey Nkey

replaces the existing protection keys on Filename in the current UFD with the owner (Okey) and nonowner (Nkey) protection keys. Valid numbers for these keys are:

Ø No Access allowed
1 Read Access only
2 Write Access only
3 Read and Write Access
4 Delete/Truncate only
5 Delete/Truncate and Read
6 Delete/Truncate and Write
7 All Access allowed (Read/Write/Delete/Truncate)

The owner can restrict his own access to a file by the PROTECtion mechanism, which can be useful in preventing accidental deletion or overwriting by an owner of an important file. A nonowner cannot give the PROTECT command and achieve desired results. The command will return the message: NO RIGHT and terminate.

A user obtains owner status to a UFD by attaching to the UFD, giving its name and owner password in the ATTACH command (refer to The Interactive Users' Guide.) A user obtains nonowner status to a UFD by giving its name and nonowner password in the ATTACH command.

A user can find out his owner status through the LISTF command. LISTF types the name of the current UFD, its logical device, O, if the user is an owner, or N if the user is a nonowner. LISTF then types the names of all files in the current UFD. An owner can determine the protection keys on all files in the current UFD through use of the file utility, FUTIL, (refer to Section 4).

Other Features of File Access

The owner/nonowner status is updated on every ATTACH and separately maintained for the current UFD and home UFD.

A user's privileges to files under a segment directory are the same as his privileges with the segment directory.

The protection keys of a newly created file are:

owner has all rights

nonowner has none

The passwords of a newly created UFD are:

owner password is blank

nonowner password is zero (any password will match)

A nonowner cannot create a new file in a UFD, or successfully give the CNAME, PASSWD, or PROTECT commands.

Furthermore, a nonowner cannot open his current UFD for reading or writing (refer to Section 3).

In the context of file access control, the MFD has all the features of a UFD. Therefore, an MFD can be assigned owner/nonowner passwords, and the UFD's subordinate to the MFD may have their access controlled by protection keys, via the PROTECT command.

If file access is violated, the error message is:

NO RIGHT

PRIMOS II File Access Control

The PRIMOS II operating system does not have file access control over individual files, but it is compatible to a degree with PRIMOS III and IV. Under PRIMOS II, a user cannot obtain access to a UFD by ATTACHing with the nonowner password. If the owner password has been given, the ATTACH is successful, but subsequent access to files in the directory is not checked. Files created under PRIMOS II are generated with the same protection keys as under PRIMOS III and IV. The passwords of a newly created UFD are the same as under PRIMOS III and IV.

File Data Access Methods

Under PRIMOS, the means of file access is the Sequential Access Method (SAM) or the Direct Access Method (DAM). With both methods, the file appears as a linear array of words indexed by a current position pointer. The user may read or write a number of words beginning at the pointer, which is advanced as the data is transferred. A file I/O module service call (PRWFIL) provides the ability to position the pointer anywhere within an open file. File data can be transferred anywhere in the addressing range. When a file is closed and re-opened, the pointer is automatically returned to the beginning of the file. The pointer can be controlled by both the FORTRAN REWIND statement and PRWFIL positioning.

With the DAM method of access, the file also appears to be a linear array of words, but this method has faster access times in positioning commands. PRIMOS keep an index that allows for positioning of the first 440 disk records of a file (1024 records of a storage module disk).

OPERATING SYSTEM USER INTERACTION

Loading and Initializing PRIMOS II

The PRIMOS II monitor is a saved-memory-image file under the UFD DOS. It must be loaded into the high-speed memory with the aid of a bootstrap loading program. The bootstrap is loaded on the devices available under control of PROM, which is located on the control panel. A system with full disk bootstrap microcode can load PRIMOS II directly from the master disk through the panel LOAD function; Other configurations may require a key-in loader and paper-tape bootstrap. For information on this and other operating procedures, see the Computer Room

User Guide.

PRIMOS internal and external commands are described in The Interactive Users' Guide (MAN 26Ø2). Prime system programs (compiler, assembler, editor, etc.) requiring detailed operating instructions are described in the pertinent user guides.

Startup

When PRIMOS II is loaded and started, it prints the message OK: on the terminal as a cue that is ready to receive commands. The first command a PRIMOS II user enters must be a STARTUP. This command assigns logical disk numbers to the physical disk drives in the particular system. The STARTUP command determines which disk surface is accessed for MFD and other command functions, and determines the order in which PRIMOS II searches disk surfaces for UFD's. Use of the STARTUP command is discussed in greater detail in the Computer Room User Guide (MAN 26Ø3).

Commands

PRIMOS commands fall into two major categories: the internal commands (implemented by subroutines that are memory-resident as part of PRIMOS) and external commands (executed by programs saved as disk files in the command UFD, CMDNCØ).

On receiving a command at the system terminal, PRIMOS checks whether it is an internal command, and if so, executes it immediately. Otherwise, PRIMOS looks in the command directory of Logical Disk Unit Ø for a file of that name. If the file is found, PRIMOS RESUMEs the file (loads it into memory and starts execution). All files in the command directory are SAVEd memory image files, ready for execution. Most are set up to return automatically to PRIMOS when their function is complete or errors occur. The command line that caused the execution of the saved program is retained and may be referenced by the program to obtain parameters, options, and filenames. To add new external commands, the user simply files a memory image program (SAVEd file) under the command directory UFD (CMDNCØ). Memory image files may also be kept in other directories and executed by the RESUME command.

File Input/Output

With the aid of the PRIMOS subroutine PRWFIL, the user can bypass formatted FORTRAN I/O and write directly from memory arrays to disk files, as in:

```
CALL PRWFIL  (1,1 PTEXT, 36, Ø, Ø)
```

This statement reads 36 words from the file associated with Funit 1 to memory array TEXT, where PTEXT is a pointer to the beginning of array TEXT. PTEXT may be set up by a call to the FORTRAN function LOC. The statement to set up PTEXT would be:

PTEXT = LOC (TEXT)

Command Files

As an alternative to entering commands one at a time at the terminal, the user can transfer control to a command file by the command: COMINPUT. This command switches command input control from the terminal to the specified file. All subsequent commands are read from the file. One can assign any unit for the COMINPUT file; i.e., command files may call other command files. For detailed information on the COMINPUT command, refer to the Interactive Users' Guide (MAN 2602).

Command files are primarily useful for performing a complicated series of commands repeatedly, such as loading an extensive system in the debugging stages (when it is necessary to recompile and reload often). Command files are also useful in system building when many files must be assembled, concatenated, loaded, etc., (for example, configuring an RTOS system or generating library files).

Saving Programs

After compiling or assembling a program and loading the object version along with library routines, the user can save the program development efforts by the SAVE command:

    SAVE Filename SA EA PC A B X Keys

This command string assigns a file, Filename, in the current UFD, saves the memory image between limits SA (starting address)0 and EA (ending address), and enters other parameters into the header block:

PC	Program Counter setting (address at which to start program execution)
A	A Register value
B	B Register value
X	X (Index) Register value
Keys	Status keys (as processed by INK, OTK instructions)

The preferred way to save newly loaded programs is to use the loader's SAVE command. Refer to the Program Development Software User Guide (MAN 1879) for details.

When a program is restored to operation by a RESTORE or RESUME command, these RVEC parameters are retrieved with the file and replaced in the registers from which they were obtained. For more detailed information, see the Interactive Users' Guide.

A program saved in the command UFD (CMDNC0, for example) can be in-

voked by name like any other external PRIMOS command. All standard Prime translator and utility programs are supplied in this form.

File Maintenance (FIXRAT)

To give the user an efficient and thorough way to check the integrity of data on a PRIMOS disk, PRIMOS provides a file maintenance program, FIXRAT, filed under the command directory, CMDNC0. When FIXRAT is invoked as an external command, it checks for self-consistency in the structure of pointers in every record, file, and directory on the disk. If there are breaks in the continuity of double-strung pointers, discrepancies between the DSKRAT file and the reconstructed record availability map, or other error conditions, FIXRAT prints appropriate error messages. FIXRAT asks the user to specify whether or not to take certain steps to repair a damaged file structure on a particular logical disk unit. For details and examples, refer to FIXRAT description in the Computer Room Users' Guide.

SECTION 3

FILE SYSTEM SUBROUTINES

INTRODUCTION

PRIMOS provides the user with a powerful and general file system. The key definitions of file system library subroutines SEARCH, PRWFIL, and ATTACH are complicated. To keep things straight, the definitions of these file system subroutines have been written with mnemonic keys (refer to Appendix C). This section describes subroutines that may be used under PRIMOS.

CALLING AND LOADING LIBRARY SUBROUTINES

When a FORTRAN user calls a subroutine, a call to the required subroutine is automatically inserted in the FORTRAN object program by the compiler.

After a FORTRAN or Macro Assembler main program is loaded, library subroutines are loaded by using the loader´s LIB (or LI) command.

CALLING SEQUENCE NOTATION

The following conventions apply to the FORTRAN calling sequence formats described in the rest of this section. For assembly language calling, refer to the PMA manual.

Items in capital letters are to be reproduced literally. Items in initial caps are variables to be assigned names or values by the user. For example, the calling sequence:

```
CALL CMREAD (Array)
```

means that the user must enter CALL CMREAD, as specified, but may coin his own array name. Common abbreviations such as Funit, Ldisk, etc. are defined in the Interactive User Guide (MAN 2602).

File names and UFD names used in routines such as ATTACH, RESTOR, etc., may be specified either by a Hollerith string or an array name. The Hollerith form allows the file or UFD name to be expressed literally in a 6-character Hollerith string such as 6HFILNAM. If an array name is used instead, it must designate a 3-word integer array that contains the file or UFD name. For example, the user could specify an array NAM that contains filename FILNAM in the following form:

```
NAM(1) FI
NAM(2) LN
NAM(3) AM
```

In either the Hollerith or Array form, the name must be specified as exactly six characters; if the name has fewer than six characters, it must be left-justified and the Hollerith string (or array) filled with space characters (represented by "b" in the example below). For example, the filename FIL1 must be treated as follows:

```
6HFIL1bb or NAM(1) or FIL1 = FI
            NAM(2)         = L1
            NAM(3)         = bb
```

Numerical values such as Funit, Ldisk, etc. must be specified by decimal integer expressions. The error return Altrtn must be set by an ASSIGN statement to the value of a statement number within the user's program. (The form $n, where n is the statement number, is also acceptable.)

Example:

The ATTACH subroutine has the general form:

```
CALL ATTACH (Ufd, Ldisk, Password, Key, Altrtn)
```

The user might code a call to this subroutine as follows:

```
CALL ATTACH (6HUSER1 , 0, PWD, 1, $50)
```

where:

1. 6HUSER1, literally identifies the user's UFD, "USER1 ",
2. The USER1 UFD is on logical disk unit 0 (Ldisk is an integer),
3. The user stores his current password in 3-word integer array PWD,
4. The variable KEY (declared as integer mode in the user's program) controls the way that the file is referenced and the home UFD set up,
5. In case of uncorrectable error, control passes to statement label 50 in the user's program.

FILE SYSTEM AND TERMINAL I/O SUBROUTINES

PRIMOS provides subroutines that simplify disk input/output, permit user programs to communicate with the PRIMOS supervisor and file structure, and provide various input/output and control functions. The

subroutines SAVE, RESTOR, RESUME, and ATTACH have the same effect as the commands of the same name, but they are called from, and return control to, a user program. The calling sequence provides the parameters that are normally entered from the terminal. Most routines, like SEARCH, SAVE, and RESTOR are implemented by code within PRIMOS II itself. A small interlude program executes a supervisor call to PRIMOS to do the work in each case.

Subroutines from this group are loaded by users from the main library file FTNLIB if they are called in a user's FORTRAN program. The subroutines described in this section in alphabetical order are:

ATTACH	ERRSET	RECYCL	T$AMLC
BREAK$	EXIT	RESTOR	T$SLC
CMREAD	FORCEW	RESUME	UPDATE
COMANL	GETERR	RREC	WREC
COMINP	GINFO	SAVE	
D$INIT	PRERR	SEARCH	
DUPLX$	PRWFIL	TIMDAT	

Other subroutines such as file, device, and terminal input-output subroutines are described in detail in the Subroutine Library Manual (MAN 1880).

****
* ATTACH *
****

The ATTACH subroutine has the same effect as the ATTACH internal command. The calling sequence is:

CALL ATTACH (Ufd, Ldisk, Password, Key, Altrtn)

To access files, the file system must be attached to some User File Directory (UFD). This implies that the file system has been supplied with the proper file directory name and either the owner or nonowner password, and the file system has found and saved the name and location of the file directory. After a successful attach, the name, location and owner/nonowner status of the UFD is referred to as the current UFD. As an option, this information may be copied to another place in the system, referred to as the home UFD. The ATTACH subroutine does not change the home UFD unless the user specifies to change it in the subroutine call. The user gets owner status if he gives the owner password, or gets nonowner status if he gives the nonowner password. The owner of a file directory can declare on a per-file basis what rights a nonowner has over the owner´s files. The nonowner password may be given only under PRIMOS III and IV. (Refer to the description of the commands PASSWD and PROTECT of the Interactive Users´ Guide (MAN 26Ø2) for more information.)

In attaching to a directory, the subroutine ATTACH specifies where to look for the directory. ATTACH specifies a User File Directory (UFD) in the Master File Directory (MFD) on a particular logical disk, a sub-directory in the current UFD, or the home UFD as the target-directory of the ATTACH operation. The format is:

CALL ATTACH (NAME, LDISK, PASSWORD, KEY, ALTRTN)

KEY is composed of two subkeys that are combined additively: REFERENCE and SETHOME. All calls require a REFERENCE subkey. The REFERENCE subkeys are shown in the following table:

REFERENCE	Octal Value	Meaning
MFDUFD	0	Attach to NAME in MFD on LDISK
CURUFD	2	Attach to NAME in current UFD

The SETHOME subkeys are required on call; these subkeys are shown in the following table:

SETHOME	Octal Value	Meaning
---	0	Do not set home UFD to current UFD after attaching.
SETHOM	1	Set home UFD to current UFD after attaching.

The meaning of the remaining parameters on a call to ATTACH is as follows:

NAME — If the key is 0 and NAME is 0, the home UFD is attached.

If the reference subkey is MFDUFD or CURUFD, NAME is either a six-character Hollerith expression or the name of a three-word array that specifies a Ufdname to be attached.

LDISK — If the reference subkey is MFDUFD, LDISK is the logical disk on which the MFD is to be searched for UFD NAME. LDISK must be a logical disk that has been started up by the STARTUP command. The special LDISK octal code 100000 signifies: search all started-up logical devices in order 0, 1, 2 ... n and attach to the UFD in which NAME appears in the MFD of the lowest numbered logical device. The special LDISK octal code 177777 signifies: search the MFD of the Ldisk currently attached to NAME.

If the reference subkey is CURUFD, or NAME is 0, LDISK is ignored and is usually specified as 0.

PASSWORD — If the reference subkey is MFDUFD, CURUFD, or SEGUFD, PASSWORD is either a six-character Hollerith expression or the name of a three-word array that specifies one of the passwords of UFD NAME. If the password is blank, it is specified as three words of two blank characters.

ALTRTN An integer variable assigned the value of a label in the user's FORTRAN program, to be used as an alternate return in case of error. If this argument is 0 or omitted, an error message is printed and control returns to PRIMOS II or III.

A UFD attached through a segment directory reference does not have a name. On LISTF, such a UFD is listed with a name of six asterisks.

If an error is encountered and control goes to Altrtn, ERRVEC(1), a PRIMOS II vector, is set to the error type as follows:

Code	Message
AH	Name NOT FOUND
AL	No UFD ATTACHED
AR	Not a UFD (detected by PRIMOS III only)

A user obtains ERRVEC through a call to GETERR. The error 'Name NOT FOUND' is printed if one of the following errors occur:

1. KEY bad.
2. NAME is not found in the specified directory.
3. LDISK is out of range or not started up.
4. In a segment directory reference, NAME (1) is a closed unit or the unit is at end of file.

If the error BAD PASSWORD is obtained, the alternate return is never taken, and both the home UFD and current UFD are set to 0 to indicate that no UFD is attached. This feature is a system security measure to prevent a user from writing a program to try all possible passwords on a UFD.

Examples of ATTACH:

```
CALL ATTACH ('JHNDOE', -1, 'JJJ', 0, ERR)
```

Searches for the UFD, JHNDOE, in the MFD (as specified in the Key) on the current logical device. If JHNDOE is found and the password, JJJ, matches the recorded password, then UFD JHNDOE is attached. The current UFD (now JHNDOE) is not set as the home UFD (as specified in the Key). The PRIMOS vector that points to the current UFD is set to this new directory.

```
**********
* BREAK$ *
**********
```

The calling sequence is:

```
CALL BREAK$ (.TRUE.)

CALL BREAK$ (.FALSE.)
```

Under PRIMOS III or IV, the BREAK$ routine, called with argument .TRUE:

```
CALL BREAK$ (.TRUE.)
```

inhibits the CTL-P or BREAK key from interrupting a running program.

```
CALL BREAK$ (.FALSE.)
```

enables the CTL-P or BREAK characters to interrupt a running program. The LOGIN command initializes the user terminal so that the CTL-P or BREAK keys cause interrupt.

Under PRIMOS II, BREAK$ has no effect.

```
**********
* CMREAD *
**********
```

The calling sequence is:

CALL CMREAD (Array)

CMREAD reads 18 words (which represent the last command line input by the user) into the system vector ARRAY, as follows:

```
Array(1)        Command(or spaces)
Array(2)

Array(3)
     (4)        Name1(or spaces)
     (5)
     (6)
     (7)        Name2(or spaces)
     (8)
     (9)
     (10)       Par1(or zero)
     (11)       Par2(or zero)


...


Array(18)       Par9(or zero)
```

The command line may be accessed directly from ARRAY. The 'Name's are normally only UFD's or filenames and the 'Par's are octal numbers.

The last command line that has been input by the user is replaced by a new line of input by a call to the subroutines: COMANL, CNIN$ or T$AMLC. If none of these subroutines have been called before the CMREAD call, then CMREAD reads the last command line typed by the user or the last command line obtained through a command file.

```
***********
* CNAME$  *
***********
```

The CNAME routine allows the same action at user program level as the CNAME command allows at command level.

The calling sequence is:

```
CALL CNAME (Oldnam, Newnam, Altrtn)
```

CNAME changes the name of Oldnam in the current UFD to Newnam. The user must have owner status to the UFD. The arguments are:

Oldnam	A filename to be changed
Newnam	The new filename for Oldnam
Altrtn	If not Ø, control goes to Altrtn if any error occurs. If Ø, an error message is printed and control returns to PRIMOS III.

If an error is encountered and control goes to Altrtn, ERRVEC(1) is set to the error type as follows:

Code	Message
CA	Newnam BAD NAME
CZ	Newnam DUPLICATE NAME
SH	Oldnam NOT FOUND
SI	Oldnam IN USE
SL	NO UFD ATTACHED
SX	Oldnam NO RIGHT

CNAME does not run on PRIMOS II, only PRIMOS III.

****
* COMANL *
****

COMANL causes a line of text to be read from the terminal or from a command file, depending upon the source of the command stream. The line is read into a supervisor text buffer. This buffer may be accessed by a call to CMREAD. The line must be in the format of a PRIMOS command line (i.e., one to three names followed by zero to nine octal parameters).

The calling sequence is:

```
CALL COMANL
```

Example:

```
CALL COMANL

CALL CMREAD (ARRAY1)
```

****
* COMINP *
****

The COMINP routine allows the user to perform the same action at program level as the user command COMINPUT allows at command level. Refer to the Interactive User Guide (MAN 2602) for details of the COMINPUT command. Briefly. COMINP causes PRIMOS to read input from a file rather than a terminal.

The calling sequence is:

```
CALL COMINP (Name, Funit, Altrtn)
```

The arguments are:

Name	Either a three-word array containing the filename of a command file, or the words TTY, CONTIN, or PAUSE.
Funit	A File Unit (range 1 to 16; 1 to 15 under PRIMOS II) that is to be used for reading the command file.
Altrtn	If not 0, control goes to Altrtn in the event of an error while opening Name. If 0, an error message is printed and control returns to the operating system in the event of an error while opening Name.

If an error is encountered and control goes to Altrtn, ERRVEC(1) is set to the error type as follows:

Code	Message
SD	UNIT NOT OPEN
SH	Name NOT FOUND
SI	Name IN USE
SI	UNIT IN USE
SL	NO UFD ATTACHED
SX	Name NO RIGHT

A user obtains ERRVEC through a call to GETERR.

```
**********
* D$INIT *
**********
```

The D$INIT routine is called to initialize disk devices.

The calling sequence is:

CALL D$INIT (Pdisk)

when Pdisk is the physical disk number to be initialized. D$INIT initializes the disk controller and performs a seek to cylinder 0 on Pdisk. D$INIT must be called prior to any RREC or WREC calls. Pdisk must be assigned by the PRIMOS III and IV ASSIGN command before calling this routine. D$INIT is normally used only by system utilities such as FIXRAT, COPY, and MAKE.

```
**********
* DUPLX$ *
**********
```

The DUPLX$ subroutine is called to control the manner in which the operating system treats the user terminal.

The calling sequence is:

CALL DUPLX$ (Mode)

where Mode has the following meanings:

Value (Octal)	Meaning
000000	Treat terminal as full duplex
100000	Treat terminal as half duplex, perform auto-line-feed after carriage return
140000	Treat terminal as half duplex

DUPLEX$ has no effect under PRIMOS II.

The mode of a user terminal is not affected by the LOGIN or LOGOUT commands.

The mode of the user terminal may also be set at the supervisor terminal by using the AMLC command.

****
* ERRSET *
****

ERRSET sets ERRVEC, a system vector, then takes an alternate return or prints the message stored in ERRVEC and returns control to the system. ERRSET has three forms:

```
CALL ERRSET (Altval, Altrtn)                         (Form 1)
CALL ERRSET (Altval, Altrtn, Mesag, Num)             (Form 2)
CALL ERRSET (Altval, Altrtn, Name, Messag, Num)      (Form 3)
```

In Form 1, Altval must have value 1ØØØØØ octal and Altrtn specifies where control is to pass. If Altrtn is Ø, the message stored in ERRVEC is printed and control returns to the system. Forms 2 and 3 are similar; therefore, the arguments are described collectively as follows:

Altval	A two-word array that contains an error code that replaces ERRVEC(1) and ERRVEC(2). Altval(1) must not be equal to 1ØØØØØ octal.
Altrtn	If Altrtn is nonzero, control goes to Altrtn. If Altrtn is zero, the message stored in ERRVEC is printed and control returns to PRIMOS.
Name	The name of a three-word array containing a six-letter word. This name replaces ERRVEC(3), ERRVEC(4), and ERRVEC(5). If Name is not an argument in the call, ERRVEC(3) is set to Ø.
Message	An array of characters stored two per word. A pointer to this message is placed in ERRVEC(7).
Num	The number of characters in Message. Num replaces ERRVEC(8).

If a message is to be printed; first, six characters starting at ERRVEC(3) are printed at the terminal. Then the operating system checks to determine the number of characters to be printed. This information is contained in ERRVEC(8). The message to be printed is pointed to by ERRVEC(7). The operating system only prints the number of characters from the message (pointed to by ERRVEC(7)) that are indicated in ERRVEC(8). If ERRVEC(3) is Ø, only the message pointed to by ERRVEC(7) is printed. The message stored in ERRVEC may also be printed by the PRERR command or the PRERR subroutine. The contents of ERRVEC may be obtained by calling subroutine GETERR.

```
**********
*  EXIT  *
**********
```

The EXIT subroutine provides a way to return from a user program to PRIMOS; it prints OK; (or OK,) at the terminal and resumes control.

The calling sequence is:

```
CALL EXIT
```

The user may open or close files or switch directories, and restart a FORTRAN program at the next statement by typing S (i.e., START).

```
**********
* FORCEW *
**********
```

The calling sequence is:

```
CALL FORCEW (0, Funit)
```

The FORCEW subroutine, under PRIMOS III and IV, immediately updates to the disk the file that is currently open on Funit. Normally this action is not needed, since the system automatically updates all changed file system information to the disk at least once per minute. Under PRIMOS II, the FORCEW routine acts as a no-operation (i.e., it does nothing).

****
* GETERR *
****

A user obtains ERRVEC contents through a call to GETERR.

The calling sequence is:

```
CALL GETERR (Xervec, n)
```

GETERR moves n words from ERRVEC into Xervec.

On an alternate return:

Error code
ERRVEC(1)

Alternate value
ERRVEC(2)

On a normal return:

PRWFIL:
ERRVEC(3) Record number
ERRVEC(4) Word number

Key of read/write convenient:
ERRVEC(2) No. of words transferred

SEARCH:
ERRVEC(2) File type

```
**********
* GINFO  *
**********
```

The calling sequence is:

CALL GINFO (Xervec, n)

GINFO moves n words in Xervec.

The information acquired is:

Information for PRIMOS II:

Xervec Word	Content
1	Low bound of PRIMOS II and buffers (77777 octal of 64K PRIMOS II).
2	High bound of PRIMOS II (77777 octal of 64K PRIMOS II).
3	(not valid)
4	(not valid)
5	Low bound of PRIMOS II and buffer (64K PRIMOS II only).
6	High bound of 64K PRIMOS II.

Information for PRIMOS III and IV.

Xervec Word	Content
1	0
2	0
3-6	(not valid)

```
**********
* PRERR  *
**********
```

PRERR prints an error message on the user's terminal.

The calling sequence is:

    CALL PRERR

Example of Use

A user wants to retain control on a request to open a unit for reading if the name was not found by SEARCH. To accomplish this, the user calls SEARCH and gets an alternate return. He then calls to GETERR and determines if another type of error occurred than NAME NOT FOUND. To print the error message while maintaining program control, the user calls PRERR.

```
**********
* PRWFIL *
**********
```

Definition of PRWFIL

PRWFIL is used to read, write, and position a file open on a file unit. A typical call to PRWFIL will read into a user buffer N words from a file open on Funit, starting at the file pointer in the file. A user may instead move the file pointer to an absolute position in the file. The two operations of reading-and-positioning or writing-and-positioning may be combined into a single call, with position occurring either before or after the read or write operation.

The calling sequence is:

```
CALL PRWFIL (KEY, FUNIT, LOC (BUFFER), NWORDS, POSITION, ALTRTN)
```

KEY is composed of three subkeys that are combined additively: RWKEY, POSKEY, and MODE. The POSKEY is required only on those calls in which positioning is requested. Subkeys with values of 0 may be omitted from the call. The PRWFIL call may be represented as:

```
CALL PRWFIL (RWKEY+POSKEY+MODE,FUNIT,PBUFFER,NWORDS,POSITION,ALTRTN)
```

The RWKEY subkeys are shown in the following table:

RWKEY	Octal Value	Meaning
PREAD	1	Reads NWORDS from FUNIT into BUFFER
PWRITE	2	Write NWORDS from BUFFER

The POSKEY subkeys are shown in the following table:

POSKEY	Octal Value	Meaning
PREREL	0	Moves the file pointer of FUNIT POSITION words relative to the current position before reading or writing
POSREL	20	Moves the file pointer of FUNIT POSITION words relative to the current position after reading or writing
PREABS	10	Moves the file pointer of FUNIT to an absolute position specified by POSITION (1) and POSITION(2) before reading or writing
POSABS	30	Moves the file pointer of FUNIT to an absolute position specified by POSITION

(1) and POSITION(2) after reading and writing

The MODE subkeys are shown in the following table:

MODE	Octal Value	Meaning
--	Ø	Reads or writes NWORDS
PCONV	4ØØ	Reads or writes a convenient number of words; less than or equal to NWORDS

The meaning of the remaining parameters in a call to PRWFIL are as follows:

FUNIT — A file unit number 1 to 16 for PRIMOS III and IV (1 to 15 for PRIMOS II) upon which a file has been opened by a call to SEARCH or a command. PRWFIL actions are performed on this file unit.

BUFFER — Reading or writing is initiated at BUFFER. Note that BUFFER is obtained through the integer function LOC.

NWORDS — If the mode subkey is Ø, NWORDS is the number of words to be transferred to or from a file unit and a user buffer. If NWORDS is Ø, no words are transferred.

If the MODE subkey is PCONV, NWORDS is the maximum number of words to be transferred. The number actually transferred is a number between 1 and NWORDS that is convenient and fast for PRWFIL to transfer. If NWORDS is Ø, no words are transferred. The user can establish how many words were transferred from ERRVEC(2).

For either mode, NWORDS may be between Ø and 65535.

POSITION — If the POSKEY is PREREL or POSREL, POSITION is a single signed integer word for relative positioning. Positioning is forward and backward from the file pointer, depending on the POSITION sign. If POSITION is Ø, no positioning is done.

If the key is PREABS or POSABS, POSITION is a two-word integer array (V-record-number, word-number) for absolute positioning. If POSITION is (Ø,Ø) (both values Ø), the file pointer is moved to the beginning of the file.

ALTRTN An integer variable assigned the value of a label in the user's FORTRAN program to be used as an alternate return in case of uncorrectable errors. If the argument is 0 or omitted, an error message is printed and control returns to PRIMOS.

If an error is encountered and control goes to ALTRTN, ERRVEC(1) is set to the error type. This is a two-character code as follows:

Code	Message	Meaning
PD	PRWFIL UNIT NOT OPEN	Bad key, or file unit not open for read/write
PE	PRWFIL EOF	End-of-file reached on read or position
PG	PRWFIL BOF	Beginning of file reached on read or position
DJ	DISK FULL	No room left on disk

A user obtains ERRVEC through a call to GETERR, which is described in this section. A user may wish to handle one type of error and have the system type all other error messages and return to PRIMOS II or III. The user can tell PRERR to print the error message that would have been printed without ALTRTN.

On a PRWFIL EOF or PRWFIL BOF error, ERRVEC(2), is set to the number of words left to be transferred in the read or write requests. On all normal returns from PRWFIL, ERRVEC(3) and ERRVEC(4) are set to the file pointer of the file as a two-word array (record-number, word-number). On a call with the PCONV subkey, ERRVEC(2) is set to the number of words read.

On a DISK FULL error, the file pointer is set to the value it had at the beginning of the call. The user may, therefore, delete another file and restart the program by typing START. This feature works only with PRIMOS III and IV.

During the positioning operation PRWFIL, PRIMOS maintains a file pointer for every open file. Because a file may contain more than 64,535 words, the largest unsigned integer that can be represented in a 16-bit word, the file pointer occupies two words. The method of representation chosen is two words, the first of which is the V-record number and the second of which is a word number. Each V-record contains 440 words of data so the word number has a range of 0 to 439. The V-record number has a range of 0 to 32767. When a file is opened by a call to SEARCH, the file pointer is set so that the next word read is the first word of the file. The position pointer contains V-record 0, word 0, or briefly (0,0). If the user calls PRWFIL to read 490 words and does no positioning, at the end of the read operation the file pointer is

(V-record 1, word 50) or briefly (1,50). The V-record size (440) is constant for all disks and does not correspond to the physical record size.

A call to read or write N words causes N words to be transferred to or from the file, starting at the file pointer in the file. Following a call to transfer information, the file pointer is moved to the end of the data transferred in the file. Using POSKEY of PREABS or POSABS, the user may explicitly move the file pointer to (record number, word number) before or after the data transfer operation. Using a POSKEY of PREREL or POSREL, the user may explicitly move the file pointer forward POSITION words from the current position, if POSITION is positive. Using a POSKEY of PREREL or POSREL, the user may move the file point backward POSITION words from the current position, if POSITION is negative. The maximum position that can be moved in the call is therefore plus or minus 32767 words. Positioning takes place before or after the data transfer, depending on the key. If NWORDS is 0 in any of the calls to PRWFIL, no data transfer takes place, so PRWFIL does only a pointer position operation. On normal returns from PRWFIL, ERRVEC (3) and ERRVEC (4) contain the file pointer as (record number, word number).

The MODE subkey of PRWFIL is most frequently used to transfer a specific number of words on a call to PRWFIL. In these cases, the MODE is 0 and is normally omitted in PRWFIL calls. In some cases, such as in a program to copy a file from one file directory to another, a buffer of a certain size is set aside in memory to hold information, and the file is transferred a buffer full at a time. In the latter case, the user doesn't care how many words are transferred at each call to PRWFIL, so long as the number of words is less than the size of the buffer set aside in memory.

As the user would generally prefer to run his program as fast as possible, the PCONV subkey is used to transfer NWORDS, or less in the call to PRWFIL. The number of words transferred is a number convenient to the system, and therefore speeds up program run time. The number of words actually transferred is put in ERRVEC (2). For an example of PRWFIL use in a program, refer to Appendix C.

```
**********
* RECYCL *
**********
```

The RECYCL subroutine is called under PRIMOS III to tell the system to cycle to the next user. It is a "I have nothing to do for now" call. Under PRIMOS II, RECYCL does nothing.

The calling sequence is:

```
CALL RECYCL
```

****
* RESTOR *
****

RESTOR has the same effect under program control as the RESTORE command.

The calling sequence is:

CALL RESTOR (Vect, Filename, Altrtn)

RESTOR performs the inverse of the SAVE operation. The SAVEd parameters for a Filename previously written to disk by SAVE are loaded into the nine-word array VECT. The program itself is then loaded into high-speed memory, using the starting and ending address provided by VECT (1) and VECT (2).

If an error is encountered and control goes to Alrtrn, ERRVEC(1) is set to the error type as follows:

Code	Message
SH	Name NOT FOUND
SI	UNIT IN USE
SI	Name IN USE
SL	NO UFD ATTACHED
SX	NO RIGHT
PE	PRWFIL EOF

****
* RESUME *
****

RESUME has the same effect under program control as the RESUME command

The calling sequence is:

CALL RESUME (Filename)

****
* RREC *
****

Subroutine RREC reads one disk record from a disk into a buffer in memory. Before RREC is called, the disk must be assigned by the PRIMOS III or IV ASSIGN command and D$INIT must be called to initialize the disk.

The RREC routine is normally used only by system utilities such as FIXRAT, MAKE, and COPY.

The calling sequence is:

```
CALL RREC (LOC (Buffer), Length, N, Ra, Pdisk, Altrtn)
```

where:

Buffer — is an array into which the N words from record Ra will be transferred.

Blen — is an array of dimension N giving a list of buffer lengths (number of words).

N — bits 9-16 must be 1.

bit 1 set means do current record address check,

bit 2 set means ignore checksum error,

bit 3 set means read an entire track beginning Ra into a buffer 352Ø words long, beginning at the buffer pointed to by Bptrs (1). (This feature may be used only if RREC is running under PRIMOS II and is reading a driver connected to the 4ØØ1/4ØØ2 controller and is a 32-sector pack.)

bit 4 set means format the track. This bit is only significant for storage module disks.

Ra — is the disk record address. Legal addresses depend on the size of the disk.

Size	Ra Range
Floppy disk	0-303
1.5M disk pack	0-3247
3.0M disk pack	0-6495
30M disk pack	0-64959
128K fixed-head disk	0-255
256K fixed-head disk	0-511
512K fixed-head disk	0-1023
1024K fixed-head disk	0-2047

Pdisk is the physical disk number of the disk to be read. Pdisk numbers are the same numbers available for use in the ASSIGN and STARTUP commands.

Altrtn is an integer variable in the user's program to be used as an alternate return in case of uncorrectable disk errors. If this argument is 0 or omitted, an error message is printed.

If an error is encountered and control goes to Altrtn, ERRVEC is set as follows:

Code	Message	Meaning
ERRVEC(1) = WB ERRVEC(2) = 0	On supervisor terminal: 10 times DISK RD ERROR Pdisk Ra Status On user terminal: UNRECOVERED ERROR	Disk hardware WRITE PROTECT error
ERRVEC(1) = WB ERRVEC(2) = CR	On user terminal: 10 times DISK RD ERROR Pdisk Ra Status followed by UNRECOVERED ERROR	Current record address error

See The Computer Room User Guide (MAN 2603) for a description of status error codes.

NOTES:

Length must be between 0 and 448 unless Pdisk is a storage module, in which case Length must be between 0 and 1040. If this number is not 448 and Pdisk is 20-27 (diskette), a checksum error is always generated; bypassing can be accomplished by setting N bit 2 = 1. No check is made for legality of Ra.

On a DISK NOT READY, RREC waits for the disk to become ready under PRIMOS III or IV without printing a message. Under PRIMOS II, RREC prints a single error message and waits for

the disk to become ready.

On any other read error, an error message is printed at the system terminal, followed by a seek to cylinder zero and a reread of the record. If 10 errors occur, the message UNRECOVERED ERROR is typed to the user or Altrtn is taken.

The routine is not available through the FORTRAN library.

****
* SAVE *
****

SAVE has the same effect under program control as the SAVE command.

The calling sequence is:

CALL SAVE (Vect, Filename)

The user sets up a nine-word vector VECT before calling SAVE. VECT(1) must be set to an integer which is the first location in memory to be saved, and VECT(2) must be set to the last location to be saved. The rest of the vector may be set up at the programmer's option.

		Location
VECT(3)	P Register	7
VECT(4)	A Register	1
VECT(5)	B Register	2
VECT(6)	X Register	Ø
VECT(7)	Keys	--
VECT(8)	Spare	--
VECT(9)	Spare	--

SAVE writes, to the named disk file, the nine-word vector VECT, followed by the memory image starting at VECT(1) and ending at VECT(2).

****
* SEARCH *
****

For some program examples that show the use of SEARCH, refer to Appendix C.

Definition of SEARCH

SEARCH is used to connect a file to a file unit (open a file) or disconnect a file from a file unit (close a file). After a file is connected to a unit, PRWFIL and other routines may be called, either to position the current-position pointer of a file unit (file pointer) or to transfer information to or from the file (using the file unit to reference the file).

Opening a File

On opening a file, SEARCH specifies 1) allowable operations that may be performed by PRWFIL and other routines (these operations are read only, write only, or both read and write); 2) where to look for a file or where to add the file, if the file does not already exist; and 3) whether the file is to be opened for writing only or both reading and writing. SEARCH either specifies a filename in the currently attached user file directory or a file unit number on which a segment directory is open. In the segment directory reference, the file to be opened or closed has its beginning disk address given by the word at the current position pointer of the file unit.

SEARCH Actions

On creating a new file, the user specifies to SEARCH the file type of the new file.

The subroutine SEARCH may be used to perform actions other than opening and closing a file. SEARCH may delete a file, rewind a file unit, or truncate a file.

Upon opening a file, SEARCH sets the file pointer to the beginning of the file. Subroutines PRWFIL and others cause information to be transfered to or from the file unit, starting at the file pointer. After the transfer, the pointer is moved past the data transferred. A call to SEARCH to rewind a file causes the file pointer to be set to the beginning of the file. Subsequent calls to PRWFIL and other routines cause information transfer to occur as if the file had just been opened. A call to SEARCH to truncate a file causes all information beyond the file pointer to be removed from the file. This call is useful if one is overwriting a file with less information than was originally contained in the file.

Subroutine Call

SEARCH is used as in the following call:

Format:

```
CALL SEARCH (KEY, NAME, FUNIT, ALTRTN)
```

KEY is composed of three subkeys that are combined additively: ACTION, REFERENCE, and NEWFILE. Not all subkeys are required on every call, and subkeys with values of zero can be omitted. The SEARCH call may therefore be represented as:

```
CALL SEARCH (ACTION+REFERENCE+NEWFILE, NAME, FUNIT, ALTRTN)
```

All calls require an ACTION subkey. The ACTION subkeys are shown in the following table:

ACTION	Octal Value	Meaning
OPNRED	1	Open NAME for reading on FUNIT
OPNWRT	2	Open NAME for writing on FUNIT
OPNBTH	3	Open NAME for both reading and writing on FUNIT
CLOSE	4	Close file by NAME or by FUNIT
DELETE	5	Delete file NAME
EXIST	6	Check to see if file exists.
REWIND	7	Rewind file on FUNIT
TRNCAT	10	Truncate file on FUNIT
CNGACC	1000	Change access of file to FUNIT

The REFERENCE subkeys are shown in the following table:

REFERENCE	Octal Value	Meaning
UFDREF	0	Searches for file NAME in the current user file directory (UFD) (as defined by a previous ATTACH) and perform the action in the ACTION subkey on the specified file.
SEGREF	10	Performs the action specified in the ACTION subkey on the file with the location indicated by the file pointer designated within the array NAME(1). This file unit must be an open segment directory.

Only those calls to SEARCH that reference a file in a UFD or segment directory need the reference key. Calls that reference file units do not need this key.

The following table lists the NEWFIL subkeys:

NEWFIL	Octal Value	Meaning
NTFILE	0	New threaded (SAM) file
NDFILE	2000	New directed (DAM) file
NTSEG	4000	New threaded (SAM) segment directory
NDSEG	6000	New directed (DAM) segment directory
NEWUFD	10000	New User File Directory (SAM)

Only those calls to SEARCH that generate a new file require a NEWFIL subkey. On other calls, this subkey is ignored.

The name of the remaining parameters in a call to SEARCH are as follows:

NAME If the reference subkey is UFDREF, NAME is either a six-character Hollerith expression or the name of a three-word array that specifies a filename (existing or not).

If the reference subkey is UFDREF and NAME(1) is -1, the current UFD is opened. NAME = -1 must be used only in configuration with ACTION subkeys 1, 2, or 3. Owner status of the current UFD is required.

If the reference subkey is SEGREF, NAME is a file unit(1-16; 1-15 under PRIMOS II) on which a segment directory is open.

On calls in which the ACTION key requires only a file unit to specify the file to be acted on, NAME is ignored and, usually, specified as 0.

FUNIT On calls that require a file unit number, FUNIT is a number 1 to 16 (1-15 under PRIMOS II). On calls that require no unit number, FUNIT is ignored and usually specified as 1.

ALTRTN ALTRTN is an integer variable assigned the value of a label return in the user's FORTRAN program to be used as an alternate in case of uncorrectable errors (e.g., attempting to open a file that is already open). If this argument is 0 or omitted, an error message is printed; control returns to PRIMOS if any error should occur while using SEARCH.

Error Messages

If an error is encountered and control goes to ALTRTN, ERRVEC(1) is set to a two-character code as follows:

Code	Message	Meaning
SA	BAD CALL TO SEARCH	Some parameter in call is invalid
SD	UNIT NOT OPEN	Attempt to truncate or rewind a file on a closed unit
SD	Name OPEN ON DELETE	Self-explanatory
SH	Name NOT FOUND	File Name not in UFD
SI	Name IN USE	File Name is already open
SI	UNIT IN USE	File unit is already open
SK	UFD FULL	Self-explanatory
SL	NO UFD ATTACHED	Self-explanatory
SQ	SEG-DIR ERROR	*SEG-DIR ERROR
SX	NO RIGHT	Access rights violation
DJ	DISK FULL	No room left on disk

*SEG-DIR ERROR:

Meaning

1. If attempting to open an existing file in the segment directory, *SEG-DIR ERROR means:

   a. The segment directory unit specified in NAME is not open for reading.

   b. The file pointer of the segment directory unit is at end of file, and therefore points to no disk address.

   c. The file pointer of the segment directory unit points to a 0 entry.

2. If attempting to open a new file in the current segment directory, *SEG-DIR ERROR means:

   The segment directory unit specified in NAME is not open for both reading and writing.

When a user obtains ERRVEC through a call to GETERR (described in this section), control is to go to ALTRTN. A user may wish to handle one type of error and have the system print all other error messages and return to PRIMOS. The user can call PRERR to print the error message that would have been printed without ALTRTN.

ERRVEC(2) is set to a file type on a normal return of a call to SEARCH to open a file, using action keys of OPNRED, OPNWRT, or OPNBTH. The codes are:

ERRVEC(2)	File Type
0	Threaded file (SAM)
1	Directed file (DAM)
2	Threaded segment directory (SAM)
3	Directed segment directory (DAM)
4	User File Directory (SAM)

Access Rights and Call to SEARCH

Under PRIMOS III and IV, the access rights of files are checked when a user attempts to open a file through a call to SEARCH. Under PRIMOS II, access rights are not checked.

A SEARCH call that creates a new file gives that file default access rights. Defaults access rights are: owner has all rights; nonowner has no rights (refer to Section 3 of the Interactive Users' Guide (MAN 2602) for a detailed description of access).

Adding and Deleting Files

For references to user file directories, a call to SEARCH to open a file for writing or both reading and writing causes SEARCH to look in the current User File Directory for the file. If the file is not found in the UFD, the file name and beginning disk address of a new file is appended to the UFD, and the new file is opened for the appropriate activity. Currently, UFD's are restricted to 72 files. An attempt to open a new file to a full UFD generates the message: UFD FULL. A call to delete a file from a UFD removes the name and beginning disk address from the UFD and shortens the UFD.

For references to segment directories, a call to SEARCH to open a file for writing or reading and writing causes SEARCH to examine the word at the file pointer of the referenced segment directory file unit. If the word is not zero, SEARCH considers the word to be a beginning record address of an already created file. SEARCH opens the file for writing or reading and writing. If the word is zero, SEARCH writes the beginning disk address of a new file in that word and opens the file. If the file pointer is positioned at the end of file, the file is lengthened one word and SEARCH writes the beginning disk address of a new file in that word, and opens the file. A call to delete a file from a segment directory causes the beginning disk address of a file at the file pointer of the segment directory to be replaced by zero. The segment directory is not shortened. An attempt to open a file for reading in a segment directory when its file pointer points to zero or is at end-of-file generates a SEG-DIR error. In no case is the file pointer of a segment directory moved. Generating a segment directory and filling it with files is an involved process; examples are presented in Appendices C and D.

Closing and Opening Files

On a call to close a file, SEARCH attempts to close file NAME and generates an error message or goes to the alternate return if NAME is not found. FUNIT is ignored unless NAME is 0. If NAME is 0, SEARCH ensures that FUNIT is closed. That is, it closes FUNIT if FUNIT is open but does not generate an error message if the file unit is closed.

Example:

```
CALL SEARCH  (1, 'OBJECT', 1, $50)
```

Searches for a file, OBJECT, in the current UFD and opens it for reading; if file is not found, return via statement 50 is made.

The user is allowed to open the current UFD for reading via a call to SEARCH. The calling sequence for this feature is:

```
CALL SEARCH  (1, -1, Funit, Altrtn)
```

This call opens the current UFD for reading on Funit. The user must have owner access rights to the UFD; i.e., the owner password must have been given in the most recent call to ATTACH (or ATTACH command). Control goes to Altrtn if there is no UFD attached, if Funit is already in use, or if the user does not have owner rights to the UFD.

Changing the Access of a File

A user may change the access of a file that is open on FUNIT to OPNREAD, OPNWRT, or OPNBTH.

Example:

```
CALL SEARCH (CNGACC + OPNWRT, 0, FUNIT, 0)
```

Access rights are checked to determine if the user has a right to accomplish the requested operations.

Checking the Existence of a File

If the user desires to find out if a certain file exists in the current UFD, the user can call SEARCH with the EXIST key. The file unit should be specified as 1. The file is not affected in any way and access rights are not checked.

Sharing Files

Two or more users may be attached to the same UFD at the same time. Furthermore, two or more users may have the same file open for reading, and they may be reading from the same file at the same time. File interlocks are provided to prevent one user from opening the file for reading or writing while another user has the file open for writing. File interlocks also prevent one user from opening the file for writing

while another user has the file open for reading. If these interlock situations are detected by SEARCH, the user gets the error message: FILE IN USE. The file interlocks also apply to the case of the same user attempting to open thefile on different file units (FUNITS).

****
* TIMDAT *
****

The calling sequence is:

```
CALL TIMDAT (Array, Num)
```

TIMDAT returns the date, time, CPU time, and paging time used since LOGIN, the users unique number on the system, and his login UFD name in an array as follows:

Array (1)	Two ASCII characters representing month. Example: 11
(2)	Two ASCII characters representing day. Example: 30
(3)	One ASCII character representing year followed by a space. Example: 4
(4)	Integer time in minutes since midnight.
(5)	Integer time in seconds.
(6)	Integer time in ticks.
(7)	Integer CPU time used in seconds.
(8)	Integer CPU time used in ticks.
(9)	Integer disk I/O time used in seconds.
(10)	Integer disk I/O time used in ticks.
(11)	Integer number of ticks per second.
(12)	User number.
(13)	Six-character login name, left-justified.
(14)	
(15)	Example: MSMITH

Num words of Array are set. This routine runs only under PRIMOS III and IV.

****
* T$AMLC *
****

The format is:

CALL AMLC (Protocol,Line,Config,Lword,Altrtn)

The arguments to the T$AMLC subroutine have the following significancee.

Protocol

After the system is running, users may assign the AMLC lines through the following commands:

ASSIGN AMLC [Protocol] Line [Config]

UNASSIGN AMLC Line

Line number (Ø to 17 octal for PRIMOS II, Ø to 37 octal for the 31-user PRIMOS III and IV) is assigned to the user, and I/O protocol and the line configuration word is established for Line. The following protocols are available:

TTY - terminal protocol
TTYHS - high-speed terminal protocol
TRAN - transparent protocol
TRANHS - high-speed transparent protocol
TTYNOP - disconnects terminal

Terminal protocol is used by lines controling interactive terminals. With terminal protocol, all input from the terminal is echoed if the line is set for full duplex and, in addition, a carriage return is echoed following carriage return. Bit 8 of each character is forced on. Note that .CTRL. P or BREAK does not cause the AMLC input/output program to QUIT. These characters are significant only at a PRIMOS III or IV user terminal. Both characters are ignored. A carriage return input by the terminal is transmitted as a new line to the program requesting input. Input is no longer echoed if the line input buffer becomes full. Terminal protocol is identical to that protocol used to control PRIMOS III and IV user terminals, except for the action of .CTRL. P and BREAK.

Transparent protocol is used by lines connected to peripheral devices or other computers. With transparent protocol, no input is echoed, no response is made to the input of a line feed or carriage return, and there is no transformation of carriage return to line feed.

The high-speed protocols are used by lines connected to peripheral devices that can run at greater than standard terminal speeds. These protocols are the same as those describedabove with one exceptiong: for output only, the line´s character time interrupt flag is set when

the output buffer contains more than 40 characters, and it remains set until the output buffer contains less than 40 characters. The protocols have a burst mode effect on the output device.

With a line using the high-speed protocols, a drastic increase in system overhead can result - depending upon the baud rate and the number of lines in the group. The user must be careful not to assign protocols to lines that normally have their character-time-interrupt flag always set; as, for example, the last line in each group of lines. If the protocol is not given, the transparent protocol is assigned by the operating system. The line number is specified in octal and must be equal to or less than the parameter Nuser.

Line

Line, the AMLC line number, is an octal number from 1 to 17.

Config

For other details about the Config argument refer to the Computer Room User Guide description of the AMLC command.

Lword

The optional parameter Lword is an octal parameter divided into a number of significant parts. If Bit 1 of Lword is set, the line is half duplex; if Bit 1 of Lword is reset, the line is full duplex. Bit 2 set indicates that LINE FEED is not to be echoed for CARRIAGE RETURN. Bit 2 reset indicates that LINE FEED is to be echoed for CARRIAGE RETURN. Bits 8 through 15 of Lword contain the number of the user to which the AMLC is connected. This user number is the number printed at the terminal upon LOGIN or LOGOUT, or printed by the STATUS command indicating user number. Although STATUS prints the User Number in decimal, the user must specify the User Number in octal when specifying this value in the AMLC subroutine. If the rightmost eight bits of Lword are zero, the AMLC line is not associated with any user space and is available to be assigned. Altrtn Specifies an alternate return to be taken in case of error.

The T$AMLC subroutine may be used to configure ASSIGNed AMLC lines as well as terminal AMLC lines.

```
**********
*  T$SLC *
**********
```

The driver T$SLC is available on the master disk and provides user control of a synchronous multi-line communications device.

Control

The driver is loaded in supervisor space. A user program communicates with the driver via FORTRAN-format calls to T$SLC0. The driver communicates with the user address space via buffers in the user address space specified by the user program. The data structure that is used by the driver is referred to as the control block. The control block is created by the user program in the user address space. It contains pointers to the user status buffer, to buffers that contain a message to be transmitted, or to buffers set to receive a message. The details of the data structure are summarized in the subsequent paragraphs. A special control block is required for each line.

The communications lines must be assigned to a user space before they can be used. The proper command is:

```
                | 0 |
                | 1 |
    ASSIGN SMLC | 2 |
                | 3 |
```

The ASSIGN command is given at the user terminal. One or more lines may be assigned to a user space.

Timing

The user space program runs asynchronously with message transfers. A call to T$SLC0 returns immediately after executing the control function required. The progress of the communication must be monitored by the user program through examination of the user space status buffer contents. For interpretation of the status codes, see the Prime Computer User Guide for Synchronous Multiline Controller (UG-0001, Rev. 2).

Hardware Requirements

The SMLC driver assumes the presence of a 520X Synchronous Multi-line Controller with a 5246 SMLC option. The address of the controller is 568.

Software Requirements

PRIMOS III: File T$SLC1 in UFD DVMSR3>DVMCMD>DVMBIN on Master Disk Vol. II is a PRIMOS III executable memory image file with the synchronous line controller option. It can be created by file C_LS16

(see UFD DVSRC on Master Disk Vol. II). In particular, file TMAIN (UFD DVSRC) must be assembled with the B-register set to 200048 and the modules that comprise T$SLC0 (refer to the SMLC User Guide) must be locked in memory.

There is a memory conflict among special drivers: the same memory and table entries are used by T$SLC0, the Gould Printer/Plotter code, and the digital input/output controller code.

User Level Software Responsibilities: a user address space program is given direct control of most of the functionality of the SMLC controller; therefore, the prospective user is assumed to know the User Guide. A specific limitation is that no more than four message blocks may be chained at a time in a given direction (transmit or receive) for a given line.

Controller status is collected as it is produced. This status is moved from interrupt response code buffers in the supervisor address space to spaces in user buffers at the next possible PRIMOS III cycle. However, the user bus/program does not get a chance to execute and act on the reported status until its turn in the round-robin cycle. If system usage is heavy enough, there will be excessive delay in line response by the user-level program.

All details of implementation of a communications protocol are left to the user program with one exception: the driver program automatically disables an active transmitter when the LAST CHARACTER OUT status is detected for that line.

Information in the user program's status buffer consists of all status words received from the controller plus two special codes. One is a code indicating the time at which the LAST CHARACTER OUT (LCT) status was detected by the driver interrupt code. This time is taken from VCLOKo and always inserted following a LAST CHARACTER OUT status word in the status data stream. The value can be related to the (seconds, ticks) time value obtained from a call to TIMDAT as follows:

LCT time (sec) = floor [(status time - vqutmo)/clock
LCT time (tics) = remainder [LCT time (sec)

where: vqutmo = -60 clock
clock = buf (11) of call to TIMDAT

The status time is given modulo ("one minute")

The other status code indicates that the stream of controller status data has overflowed either an internal supervisor buffer or the user program status buffer. If this is detected, status information has been lost. The status buffer overflow code is the integer -1 (supervisor buffer) or -2 (user buffer).

User Calls to the SMLC

The form of the user call to the supervisor is (in FORTRAN):

CALL T$SLC0 (Key, Line, Loc (Block), Nwds)

where:

1 < Key < 5;

0 < Line < 3;

Loc (Block) is the memory address of a buffer used in the call;

Nwds is the word count of Block.

The values and meaning for Key are as follows:

Key	Meaning
1	User control block is undefined. Status information is no longer moved to user program space, and controller state is unaltered. Requires two arguments (key, line).
2	Control block, which is defined and structured as in Table 3-1, defines an area to store status information and, optionally, a message chain for reception or transmission.
3	Buffer block contains four or five data words to be sent to the controller. These control words configure the line, set line control, define the programmable sync character and (optionally) set the internal programmable character-time clock. Refer to Table 3-1 for the block structure.
4	Buffer block contains one word to be used as the next data set control word. See "OTA 01XX" in the SMLC User Guide.
5	Buffer block contains one word which is used as the next receive/transmit enable word. See "OTA 14XX" in the SMLC User Guide. Half-duplex looping for odd-even line pairs is not allowed.

Table 3-1 Structure of SMLC Hardware Configuring Block

Word	Meaning
0	Receiver line configuration word. See "OTA 00XX" in the SMLC User Guide.
1	Transmitter line configuration word. See "OTA 00XX" in the SMLC User Guide.
2	Line control word. See "OTA 02XX" in the SMLC User Guide.
3	Synchronizing characters. See "OTA 03XX" in the SMLC User Guide.
4	Clock control constant. This word is optional. Note that this word controls the clock rate for all lines on the controller. See "OTA 17XX" in the SMLC User Guide.

****
* UPDATE *
****

The calling sequence is:

CALL UPDATE (Key Ø)

The possible value for Key is:

Value	Meaning
1	Update CUFD (current UFD); DSKRAT buffers to disk, if necessary; and undefine RAT in memory.

This call is normally used by users.

```
**********
*  WREC  *
**********
```

Subroutine WREC writes the disk record to a disk from a buffer in memory. The arguments and rules of the WREC call are identical to those of RREC except for bits 1 and 2 of N, which have no meaning on write. For a call to write a record on the diskette, the buffer length Length must be 448 words.

The calling sequence is:

CALL WREC (LOC (Buffer), Length, N, Ra, Altrtn)

The meaning of the parameters is the same as described under RREC in this section, except that the function of the command is to write rather than read the specified records. The user of WREC is responsible for being careful to write only on areas of the disk that do not contain significant user or operating system information.

An attempt to write on a write-protected disk generates the message:

```
DISK WT ERROR    Pdisk    Ra    Status
WRITE PROTECT
```

on the supervisor terminal and the message:

```
UNRECOVERED ERROR
```

at the user terminal. ERRVEC(1) will contain error code WB, unless Altrtn is taken. Other write errors are retried ten times in a manner similar to read errors (refer to RREC). This routine is not available through the FORTRAN library.

SECTION 4

FILE UTILITY COMMAND (FUTIL)

INTRODUCTION

FUTIL is a file utility command that provides commands for the user to copy, delete, and list files and directories. FUTIL has an attach command that allows attaching to subdirectories by giving a directory treename from either the MFD or the home UFD to the subdirectory. FUTIL allows operations not only with files within User File Directories (UFD's), but also files within segment directories. For complex operations, FUTIL may be run from a command file.

FILE STRUCTURE

A user should be familiar with the Prime file structure (refer to Section 2, entitled "File Structures"). Figure 4-1 is a sample file structure.

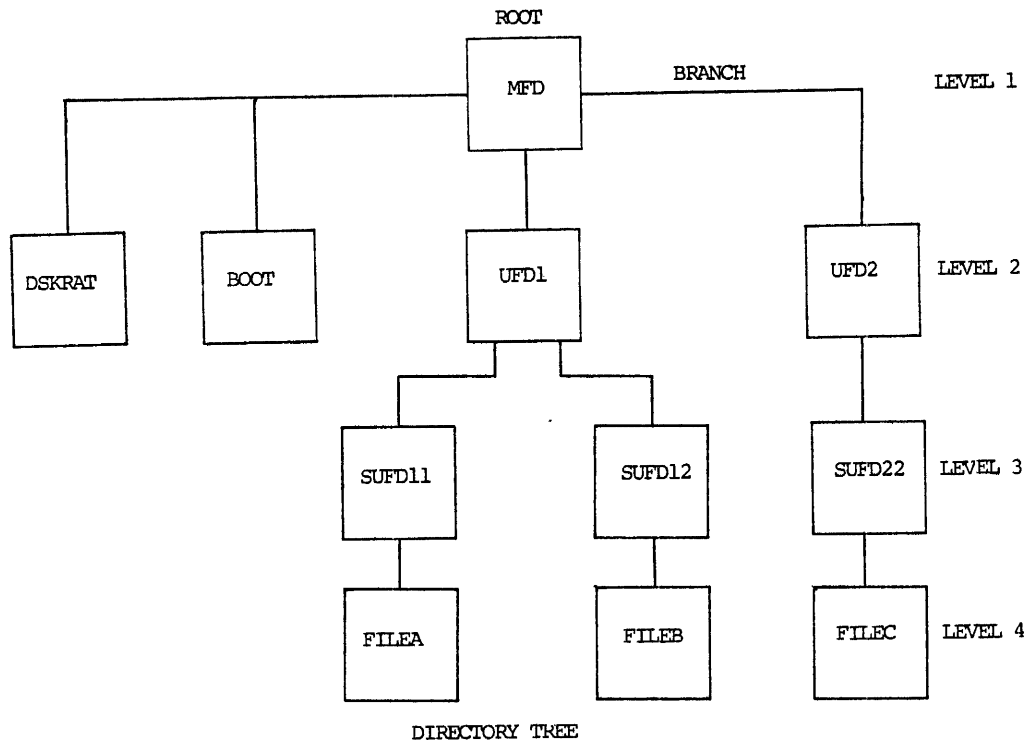


Figure 4-1. Sample File Structure

The PRIMOS file structure on any disk pack is a tree structure where the MFD is the root or trunk of the tree, the links between directories and files or subdirectories are branches, and the directories and files are nodes. A directory tree consists of all files and subdirectories that have their root in that directory. In Figure 4-1, the directory tree for UFD1 is circled. An MFD directory treename consists of a list of directories and passwords necessary to move down the tree from the MFD to any directory. For example, the MFD treename for SUFD1 is:

    MFD PASSWORD > UFD1 PASSWORD > SUFD1 PASSWORD

The character ">" separates directories in the treename and suggests that one is proceeding down a tree structure.

An MFD directory treename may optionally omit the MFD and include the logical disk number of the pack or the packname.

Examples:

    UFD1 UFD1PASSWORD > SUFD11 SUFD11PASSWORD
    < 1 > UFD1 UFD1PASSWORD > SUFD11 SUFD11PASSWORD
    < TDISK > UFD1 UFD1PASSWORD > SUFD11 SUFD11PASSWORD

The logical disk number may optionally follow the first UFD as follows:

    UFD1 UFD1PASSWORD 1 > SUFD11 SUFD11PASSWORD

If no pack name or disk number is given, the logical disk referred to is the lowest numbered logical disk in the MFD in which UFD1 appears. A user, by means of the ATTACH or PRIMOS III or IV LOGIN command, may specify a particular User File Directory in the file structure as the home UFD. Additional FUTIL ATTACH commands may refer to either the MFD or the home UFD as the starting point. If the logical disk name is specified as *, the MFD of the current logical disk is scanned for UFD1.

Example:

If the home UFD is UFD1, the home UFD pathname to SUFD11 would be:

    * > SUFD11 SUFD11PASSWORD

"*" represents the home UFD. The home UFD treename to UFD1 is simply *. This form of tree name is also referred to as a relative pathname.

A User File Directory is a file that consists of a header and a number of entries (0-72). Each entry consists of 1- to 6-character filenames, protection attributes of the file, and a disk record address pointer to the file. A segment directory is a file consisting of an unlimited number of entries, each entry being a disk record address pointer to the file. A null pointer indicates no file at that entry. To refer to a particular file in a segment directory, a user must specify the file

position of the entry in the segment directory (see Section 3 (PRWFIL) for details of file positioning). A user may specify the position as an absolute position (record number, word number) where V-record number is between 0 and 32767, and word number is between 0 and 439. As there are 440 data words in each V-record (virtual record), there are 440 files in each segment directory. The first file can be referred to as (0,0), the second as (0,1), the 440th file as (0,439), and the 441st file as (1,0). The construction (record number, word number) is referred to as a segment directory filename. In FUTIL, arguments to the commands are either User File Directory (UFD) filenames or segment directory filenames, depending on the directory type the file is under, as are names specified as parameters to the LISTF command of FUTIL.

DESCRIPTION OF FUTIL COMMANDS

To invoke FUTIL, type FUTIL. When loaded, FUTIL prints the prompt character, >, and awaits a command string from the user terminal. To terminate long operations such as LISTF, type CTRL P and restart FUTIL at 1000. A user must type a command followed by a carriage return and wait for the prompt character before using the next command. The erase character " and the kill character ? may be used to modify the command string, as in other operating system commands. In the following description of commands, underlined letters represent the abbreviation of the command or argument. [] surround optional arguments. ... means the previous element may be repeated.

```
**********
*  QUIT  *
**********
```

The format is:

QUIT

returns to PRIMOS III or IV.

```
**********
*  FROM  *
**********
```

The format is:

FROM Directory-Treename

where Directory-Treename is of the format:

<Ldisk> Directory [Password][Ldisk]> Directory [Password]...<Packname>

FROM defines the FROM directory in which files are to be searched for the commands COPY, COPYSAM, COPYDAM, DELETE, LISTF, LISTSAVE, SCAN, TRECPY, TREDEL, UFDCPY, and UFDDEL. The directory is defined from the directory treename (see format above). The treename may contain up to 10 directories that can be segment directories as well as User File Directories. If segment directories are specified, the user must have read access rights to them. If any error is encountered, the FROM directory is set to home UFD. The first directory in the treename may be *, which refers to the home UFD. The default FROM directory is the home UFD. Note that use of FROM never changes the home UFD.

The abbreviation for FROM is F.

Examples:

FROM <0> CARLSO

Set FROM directory to CARLSO on logical disk 0. CARLSO must be in the MFD on logical disk 0 and have a blank password.

FROM CARLSO ABC

Search the MFD on all started disks for CARLSO in logical disk order 0 - 17. Set the FROM directory to the first directory encountered named CARLSO. One of the passwords of CARLSO must be ABC.

FROM <TSDISK> CARLSO > SUB1 > SUB2

Set the FROM directory to SUB2. SUB2 must be a directory in SUB1; SUB1 must be a directory in CARLSO; and CARLSO must be a directory in the MFD on a disk with pack name TSDISK. The directories CARLSO, SUB1, and SUB2 must have a blank password.

FROM *

Set the FROM directory to the home UFD. The home UFD is normally the last UFD the user has logged into, or attached to with either the ATTACH or FUTIL ATTACH command. If one were logged into CARLSO, the above command sets the FROM directory effectively to CARLSO. This command does not have to be given again if the user changes the home UFD.

Furthermore, this command does not have to be given at all unless the FROM directory has been made something other than the home UFD, since the home UFD is the default.

Example:

```
FROM * > SUB1
```

Sets the FROM directory to SUB1. SUB1 must be a directory in the home UFD and have a blank password.

```
**********
*   TO   *
**********
```

The format is:

    TO Directory-Treename

TO defines the TO directory in which files are searched for the commands COPY, COPYSAM, COPYDAM, TRECPY, and UFDCPY. The treename may contain at most ten directories that may be segment directories as well as UFD's. If segment directories are specified, the user must have read, write, and delete/truncate access to them. The first directory in the treename may be the home UFD (*). The default TO directory is the home UFD. If any error is encountered, the TO directory is set to the home UFD (*).

The abbreviation for TO is T.

```
**********
* ATTACH *
**********
```

The format is:

    ATTACH Directory-Treename

ATTACH moves the home UFD to directory defined by Directory-Treename. The treename may contain at most ten directories. The first directory in the treename may be *. All directories in the treename must be UFDs.

The abbreviation for ATTACH is A.

```
**********
*  COPY  *
**********
```

The format is:

COPY FILEA [FILEB] [, FILEC [FILED]] . . .

FUTIL copies FILEA in the FROM directory to FILEB in the TO directory and optionally FILEC in the FROM directory to FILED in the TO directory. If FILEB is omitted, the new file is given the same name as the old file. FILEA and FILEC must be SAM or DAM files and cannot be directories. Read access rights are required for FILEA and FILEC. If FILEB exists prior to the copy, it must be a SAM or DAM file and the user must have read, write, and delete/truncate access rights to the target file (FILEB in this case). If FILEB exists, it is deleted; then FILEA is copied to FILEB. The file type of FILEB will be the same as FILEA.

The abbreviation for COPY is C.

Examples:

COPY FILEA

copies FILEA in the FROM directory to FILEA in the TO directory.

COPY FILEA , FILEB , FILEC

copies FILEA, FILEB, and FILEC in the FROM directory to FILEA, FILEB, and FILEC in the TO directory.

COPY FILEA FILEB

copies FILEA in FROM directory to FILEB in TO directory.

COPY FILEA1 FILEA2,FILEB1 FILEB2,FILEC1 FILEC2

copies FILEA1, FILEB1 and FILEC1 in the FROM directory to FILEA2, FILEB2, and FILEC2 in the TO directory.

COPY (Ø,Ø)

In this case, the FROM directory and TO directory must each be segment directories. FUTIL copies the file at position (Ø,Ø) of the FROM directory to position (Ø,Ø) of the TO directory. There are no access rights attached to these files, so PRIMOS III and IV check instead the access rights of the directory. A user cannot set the FROM and TO directories if they are segment directories without access rights to them. No spaces are allowed in the name (Ø,Ø).

COPY (Ø,Ø) (Ø,1)

copies the file at position (0,0) of the FROM directory to position (0,1) of the TO directory, both of which are segment directories.

****
* COPYSAM *
****

The format is:

COPYSAM FILEA [FILEB] [, FILEC [FILED]] . . .

The function is the same as COPY but COPYSAM also sets file type of FILEB and FILED to SAM, instead of copying the type of FILEA and FILEC.

The abbreviation for COPYSAM is COPYS.

****
* COPYDAM *
****

The format is:

COPYDAM FILEA [FILEB] [, FILEC FILED] . . .

The function is the same as COPYSAM, but COPYDAM sets file type of FILEB and FILED to DAM.

The abbreviation for COPYDAM is COPYD.

```
**********
* TRECPY *
**********
```

TRECPY is used to copy directory trees. A directory tree consists of all files and subdirectories that have their root in that directory.

The format is:

TRECPY DIRA [DIRB] [, DIRC [DIRD]]

This command TRECPY copies the directory tree specified by directory DIRA to directory DIRB, and optionally DIRC to DIRD. DIRB and DIRD must not exist prior to the TRECPY command. If DIRB is omitted, DIRA is taken as the name of the directory to be copied to. DIRA and DIRC must be in the FROM directory; DIRB and DIRD are created in the TO directory. Read access rights are required for DIRA and DIRC, but no access rights are required of files or subdirectories within them.

DIRB and DIRD are created with the same directory type and passwords as DIRA and DIRC, and with default access rights. The names, access rights, and passwords of all files and subdirectories are also copied.

The abbreviation for TRECPY is TR.

Example:

```
FROM MFD
TO   MFD
TRECPY   CARLSO   CARNEW
```

copies the directory tree specified by CARLSO in the MFD to a new directory, CARNEW, in the MFD.

****
* UFDCPY *
****

The format is:

UFDCPY

This command copies all files and directory trees from the FROM directory to the TO directory. The user must have owner rights in the FROM directory. Furthermore, all files and directories in the FROM directory must have read access rights. Files already existing in the TO directory with names identical to those in the FROM directory are replaced. The user must have read, write, and delete access rights to files that are to be replaced.

Directories already existing in the TO directory with names identical to those in the FROM directory will cause the copy operation to stop. Files and directories created in the TO directory will have the same file type as the old files, with the addition of default access rights. The names, access rights, and passwords of all files and subdirectories within directory trees being copied are also copied. Other existing files and directories in the TO directory are not affected unless those existing files have the same names as files in the FROM directory. In the case of duplicate names, the files in the TO directory are overwritten with the files of the same name in the FROM directory. UFDCPY is effectively a merge of two directories. Both the FROM and the TO directory must be UFD's.

Example:

```
FROM  CARLSO
TO    CARNEW
UFDCPY
```

copies all files and directories from CARLSO in the MFD to CARNEW in the MFD. Note that, unlike the example for TRECPY, the user has not specified the MFD as the FROM directory; therefore, he does not need to know the MFD password. In the example, CARNEW exists prior to the UFDCPY. With the TRECPY example, CARNEW does not previously exist.

The abbreviation for UFDCPY is U.

****
* DELETE *
****

The format is:

DELETE FILEA [FILEB] . . .

Deletes FILEA and optionally FILEB from the FROM directory. FILEA and FILEB cannot be directories. The user must have read, write, and delete access rights to each file specified. If FILEA and FILEB are in a segment directory, read, write, and delete rights are required for the FROM directory.

There is no abbreviation for DELETE.

Examples:

DELETE FILEA

DELETE FILEA FILEB FILEC FILED

****
* TREDEL *
****

The format is:

TREDEL Dira [Dirb]

This command deletes the directory tree specified by directory Dira and optionally deletes Dirb from the FROM directory. Dira and Dirb must be directories. The user must have read, write, and delete rights to the Dira and Dirb; however, read, write, and delete rights are not required for files and subdirectories nested with the Dira and Dirb. For example, if files named FILEA and FILED are in a segment directory, read, write, and delete access rights are required for the FROM directory. Note that the operating system DELETE command must not be used to delete directories because it does not free the disk space used by files within the directory for system usage. TREDEL correctly frees disk space to the system.

There is no abbreviation for TREDEL.

****
* UFDDEL *
****

The format is:

UFDDEL

This command deletes all files and directory trees (specified by directories) within the FROM directory. User must give the owner password in the FROM command and have read, write, and delete access to all files and directories within the FROM directory. These rights are not required for files and subdirectories nested within the directories in the FROM directory.

There is no abbreviation for UFDDEL.

****
* LISTF *
****

The format is:

LISTF [Level] [FIRST] [LSTFIL] [PROTEC] [SIZE] [TYPE]

This command lists the FROM directory treename, the TO directory treename, and all files and directory trees in the FROM directory at the terminal. LISTF optionally follows each filename by its protection attributes, size in disk records, and file type. If the LSTFIL option is given, the list of files is sent to a file named LSTFIL in the home UFD instead of to the terminal. At a later time, a user may print that file on a line printer. Level is a number specifying the lowest level in the FROM directory tree structure to be listed. (See Figure 4-1). The following list describes the output.

Level	Output
Ø	The FROM directory name
1	The FROM directory and all files and directories within it (level 1 directories)
2	All output at level 1 and all files and directories within level 1 directories

If the level is omitted, the default is 1.

The protection attribute of each file is printed as

Owner-Key Nonowner-Key

These keys are numbers with a range Ø-7 that have the following meanings:

Ø	No access allowed
1	Read access only
2	Write access only
3	Read and write access
4	Delete/truncate only
5	Delete/truncate and read
6	Delete/truncate and write
7	All access allowed

The possible file types are:

SAM	for SAM file
DAM	for DAM file

SEGSAM for SAM segment directory
SEGDAM for DAM segment directory
UFD for User File Directory

If the word FIRST is specified following LISTF, the first line of each file is printed following the name of the file. The first line is not printed for files with filenames beginning with B<- or *, since these files are considered to be object and memory image run files respectively. The first line of a file might contain useful information such as creation date, last update, author, or short description of the file contents.

LISTF traverses the file structure (as shown by the snaked line in Figure 4-2) generating printed messages in sequence (as shown in the circles adjoining the snaked line).

The abbreviation for LISTF is L.

When LISTF is used to produce a list of the sample file configuration shown in Figure 4-2, the output level is set to 3; with the SIZE option, the printed list appears as follows:

```
FROM-DIR = *       MFD

FROM-DIR = *       MFD
TO-DIR   =

BEGIN MFD          1

DSKRAT    1 BOOT        1

 BEGIN UFD              1
  BEGIN SUFD11          1

  FILEA       1

 END    SUFD11       2
 BEGIN  SUFD12       1

  FILEB       1

END     SUFD12       2
END     UFD1      5
BEGIN   UFD2      1
BEGIN SUFD21

  FILEC       1

 END    SUFD21       2
END     UFD2        3
END   MFD          11
```

LISTF, upon encountering a directory, prints the word BEGIN followed by

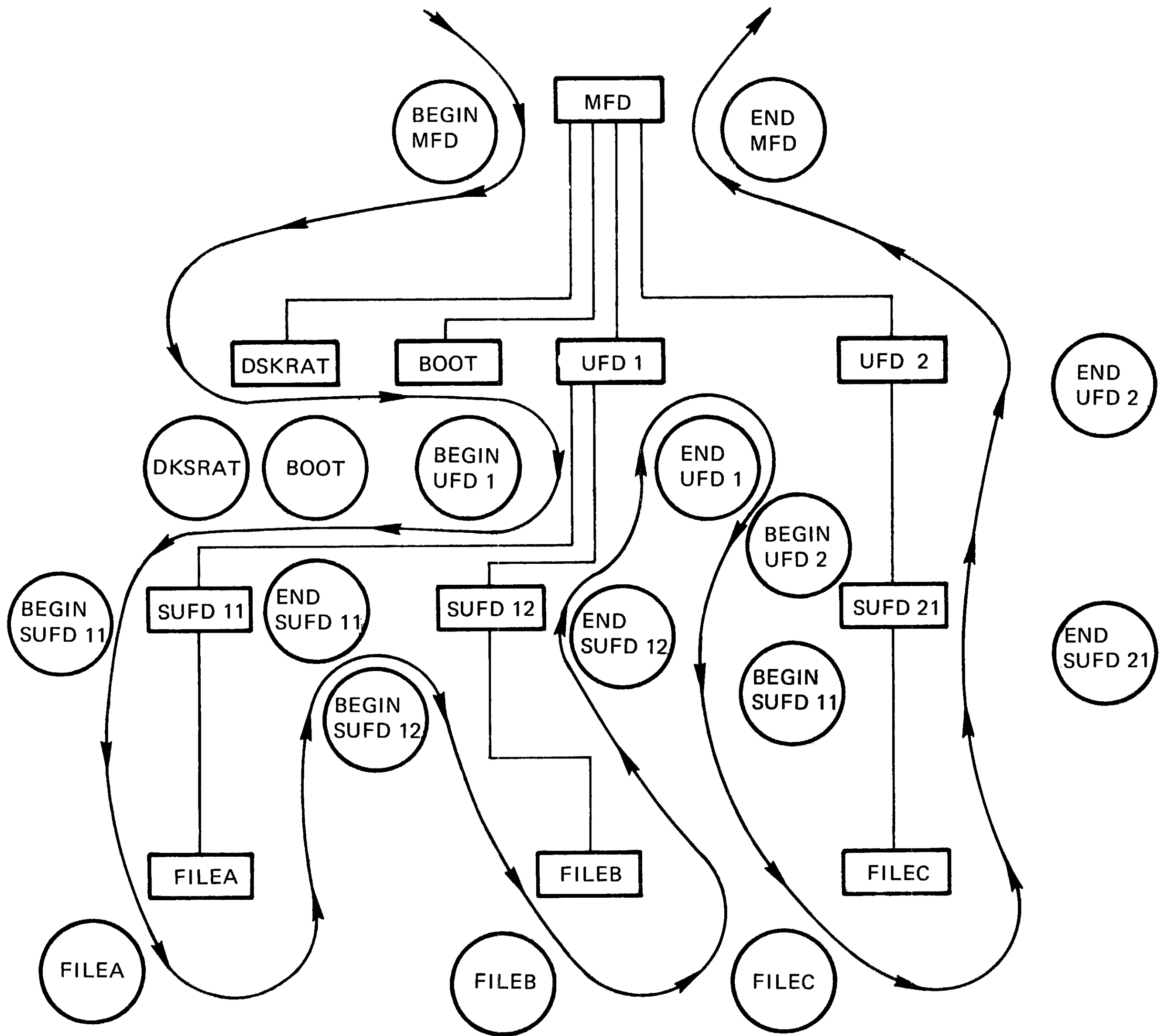


Figure 4-2. Typical Traverse of Directory Tree by FUTIL during LISTF.

the name of the directory and its size in blocks of 44 data words. On leaving a directory, LISTF prints END followed by the number of records used by all files and directories within the directory tree headed by the directory file. On encountering a file, LISTF simply prints its name and size, squeezing as many file names as will fit on each line. LISTF skips a line whenever a directory follows a file or a file follows a directory. LISTF does not count records in files lower than the specified level in the FROM directory tree.

In the above example output, the number following MFD, 11, is the total number of V-records used by the MFD directory tree and consists of all files and directories on the disk pack. LISTF indents the printed output one space for each level down in the tree in which the directory is located. This format makes it easy to understand the relationship of each directory to other directories in the tree.

A file with 0 to 440 words is considered to have one V-record; to 880 words is 2 V-records; etc.

```
************
* LISTSAVE *
************
```

The format is:

LISTSAVE Fname [Level] [PROTECT] [SIZE] [DATE] [PASSWORDS] [FIRST]

The LISTSAVE command and most of its arguments may be abbreviated as follows:

LIST Fname [Level] [P] [S] [T] [D] [PA] [F]

The LISTSAVE command is identical in function to the LSTFIL option specified, except the output listing file is named with the name specified by Fname rather than LSTFIL.

```
********
* SCAN *
********
```

The format is:

```
SCAN Fname [Level] [PROTECT] [SIZE] [TYPE]
        [DATE] [PASSWORDS] [LSTFIL] [FIST]
```

The SCAN command and most of its arguments may be abbreviated as follows:

S Fname [Level] [P] [S] [T] [D] [PA] [L] [F]

The SCAN command is used to search the FROM directory tree for the occurence of all files, subUFD's, and segment directories that are named with the name specified by Fname.

If the level specified by the argument level is 1 (the default), only the file name followed by the information specified by the optional arguments is printed. If the level specified by Level is greater than 1, the pathname (treename) to the file or directory, starting from the FROM directory, is printed. In addition, the information specified by any optional arguments may be printed after the pathname. For example, with the sample tree structure shown in Figure 4-2, the command:

SCAN FILEB S F 10

will print the following information:

```
FROM = MFD
TO = *

DIRECTORY PATH = MFD> UFD> SUFD12
   FILE B              1 = NO FIRST LINE
```

FILB lacks a first line because it was empty. Note that the name FILEB was indented three spaces because it is a sub-UFD that is a third level in a tree subordinate to the MFD.

FUTIL RESTRICTIONS

In using FUTIL under PRIMOS III or IV, certain operations may interfere with the work of other users. For example, a UFDCPY command to copy all files from a UFD currently used by another logged-in user may fail. If any file in that directory is open for writing by that user, UFDCPY encounters the error FILE ALREADY OPEN, and aborts. If the user attempts to open one of his files for writing while UFDCPY is running, the user may encounter that error. Under FUTIL, the LISTF and TRECPY commands cause the same interaction problems. (Other FUTIL commands such as COPY and DELETE can also interfere with the other user, but the problem is not so serious, since only one file is potentially involved in a conflict.) To prevent conflicts, users working together and involved in operations using each other's directories must coordinate their activities. If two users consistently use the same UFD at the same time, they must avoid the LISTF command of FUTIL, and use the system LISTF command instead.

FUTIL operations when using the MFD must be done carefully. Never give the command TREDEL MFD, because that command deletes every file on the disk except the MFD, DSKRAT, BOOT, and BADSPT. A LISTF or UFDCPY of the MFD must be done only when no other user is accessing files or directories on that disk. A UFDCPY of the MFD to the MFD of another disk has the effect of merging the contents of two disks onto one disk.

A user must be sure there is enough available space on the TO disk before attempting the LSTFIL or UFDCPY operation if FUTIL aborts. Also, the names of files and directories on the two disks may conflict. To avoid any name conflict, it may be desirable to UFDCPY the MFD of one disk into a UFD on another disk. Each directory originally on the FROM disk becomes a subdirectory in that UFD on the TO disk. For example, the contents of ten diskettes could be copied into ten User File Directories on a 1.5M word disk pack.

NOTE:

A UFDCPY of an MFD, does not copy the DSKRAT, MFD, BOOT, or BADSPT files to the TO directory.

The effect of a UFDCPY from the MFD of a disk in use to the MFD of a disk that was newly produced by the MAKE command is to reorganize the disk files so that all files are compacted; that is, files have their records close to each other on the new disk. After such a compaction, the access time to existing files on the new disk is effectively reduced from the access time on the old disk. Furthermore, new files tend to be compact, since all free disk records are also compacted. The use of a compacted disk may improve the performance of PRIMOS.

Users must not abort COPY or DELETE operations under PRIMOS II, but allow them to run to completion. Aborting a COPY or DELETE operation may cause a directory to contain incorrect entries, as for example, a file with a pointer mismatch or bad file structure, or a directory with a partial entry. Since PRIMOS does not run correctly on a directory with a partial entry, FIXRAT must be run immediately if any of the above conditions apply. Under PRIMOS III and IV, critical areas of code are surrounded by calls to BREAK$, a subroutine that inhibits the CNTRL-P key. As a result, interruption of FUTIL does not generate a bad file structure.

Error Messages

The following error messages are generated by FUTIL. In many cases, FUTIL prints error messages generated by PRIMOS and retains control, so users must be familiar with operating system error messages. The list given here includes messages that may be encountered by FUTIL. Most messages are preceded by a filename identifying the file causing the error. Some of the error messages have the format:

reason for error
FILE = filename

DIRECTORY PATH = directory treename

? = unrecognizable command

Message	Meaning
ALREADY EXISTS	An attempt has been made to TRECPY to a file that already exists or UFDCPY has attempted to copy to a directory that already exists. If you intend to do the operation, the file in the TO directory must first be deleted.
BAD NAME	A segment directory filename was given to a command that expected a UFD filename or vice versa. The type of filename must match the type of directory in which the file is contained.
BAD PASSWORD	An incorrect password has been given in a FROM, TO, or ATTACH command. As PRIMOS III and IV does not allow FUTIL to maintain control in case of a bad password, the FUTIL command must be given to restart FUTIL. The FROM directory and TO directory are reset to home UFD in this case.
BAD SYNTAX	The command line processed by FUTIL is incorrect.
CANNOT ATTACH TO SEGDIR	The last directory in the directory treename to an ATTACH command is a segment directory. It must be a UFD, because ATTACH sets the home UFD to the last directory in the path.
CANNOT DELETE MFD	User has given the UFDDEL command while attached to the MFD. This is not allowed.
DIRECTORIES NESTED TOO DEEP	Directories may be nested to a depth of 100 levels. User has attempted to exceed this limit.
DISK ERROR	May indicate a disk error or a FUTIL attempt to process a badly constructed segment directory. Running FIXRAT (refer to the Computer Room User Guide MAN 2603) is recommended.

DISK FULL	The disk has become full before FUTIL finishes a copy operation. For operations involving many files, some files are not copied, creating only partially copied directories that may be of limited use. It is suggested that the user delete such a structure immediately to prevent confusion as to what has been copied.
IN USE	Indicates a FUTIL attempt to process a file in use by some other user. It may also indicate an attempt to copy a directory to a subdirectory within itself.
IS A DIRECTORY, CANNOT COPY TO IT	Same as ALREADY EXISTS.
NO RIGHT	User has attempted an operation on a file that violates the file access rights assigned to that file. These rights may be changed by the PRIMOS III or IV PROTEC command, if the user has given the owner password on ATTACH.
NO ROOM USE DOS32	User is using FUTIL under DOS16 and has attempted an operation that causes FUTIL to run out of room. This message is not likely to occur when processsing SAM segment directories.
NO UFD ATTACHED	Self-explanatory.
NOT A DIRECTORY	User has given a directory treename that includes a file that is not a directory.
NOT FOUND	Self-explanatory.
POINTER MISMATCH	Indicates a bad file structure. Running FIXRAT is recommended.
PRWFIL EOF	User has attempted to reference a nonexistent file beyond the end of a segment directory.
SEG-DIR ER	User has attempted to reference a file in a segment directory with an entry of Ø, which indicates file does not exist or the user has attempted to

	reference a file beyond the end of the segment directory.
UFD FULL	Self-explanatory.
UNRECOVERED ERROR	Indicates either the user has attempted to write to a write-protected disk, disk error, or an attempt to process a bad file structure. Running FIXRAT is recommended if the disk was not write-protected.

APPENDIX A

FILE AND HEADER FORMATS

Table A-1. File and Header Formats

448-Word File Record Header Format

Word	Content	Remarks
0	"This" record address	Consists of the PRIMOS record address.
1	Parent Record Address or Beginning Record Address (BRA)	If record is a beginning record, this word contains a pointer to the parent (immediately superior) segment directory, or UFD.
2	Forward	Records forward pointer to the next record. (May be a null pointer if last record).
3	Backward	Records backward pointer to the immediately preceding record (may be a null pointer if first record).
4	Data Count (0-440)	Records number of words of data in this record (excludes header).
5	File Type	Only in the beginning record. 0 = SAM File 1 = DAM File 2 = SAM Segment Directory 3 = DAM Segment Directory 4 = SAM User File Directory
6	Spare 1	Reserved
7	Spare 2	Reserved

All remaining words in the record may be used to store 16-bit words. Data is assumed to continue from the ninth word in the record to the

last word of the physical record. The forward and backward pointers make it easy for PRIMOS to traverse a file in either direction, and at the same time provide a large measure of protection against snowballing disk errors. The pointer to the beginning record address makes it possible to identify a "lost" record.

1040-Word File Record Header Format

Word	Content
0	Record Address-high
1	Record Address-low
2	Beginning Record Address-high
3	Beginning Record Address-low
4	No. Data Words (0-1024)
5	File Type
6	Forward Record Address-high
7	Forward Record Address-low
8	Backward Record Address-high
9	Backward Record Address-low
10	Spare
11	Spare
12	Spare
13	Spare
14	Spare
15	Spare

NOTES:

All disks, except the storage module, have 448-word records; storage modules have 1040-word records.

The Beginning Record Address of the first record in a file points to the directory (UFD or segment directory) entry of the file. In all other records, the BRA points to the first record of the file.

The forward pointer contains the address of the next record in sequence, or contains 0 if the record is the last record in the file.

The backward pointer contains the address of the previous record, or contains 0 if the record is the first record in the file.

The File Type is valid only in the first record of the file. Possible values are:

0	SAM File
1	DAM File
2	SAM Segment Directory
3	DAM Segment Directory
4	UFD

If the file is the record Ø bootstrap (BOOT), or the DSKRAT, and the disk has 1Ø4Ø-word record size; bit 1 (1ØØØØØ octal) of the File Type will be set.

Table A-2. UFD FORMATS

Word	Content	Remarks
	UFD Header, where:	
Ø	Word Count = 8 (Size of header)	
1-3	Owner Password	Six ASCII characters
4-6	Nonowner Password	
7	Spare	
8-13	UFD Entries	See table below
14-19		

UFD ENTRY FORMAT

Word	Content	Remarks
		Word numbers are relative to beginning of entry.
Ø	BRA	Beginning Record Address of file.
1-3	Filename	Currently six ASCII characters.
4	Spaces	Reserved for future use.
5	Protection Keys	Bits 1-8 owner protection. (Inverted - Ø means all rights allowed.) Bits 9-16 non-owner protection.

Figure A-1 shows UFD file format and use schematically.

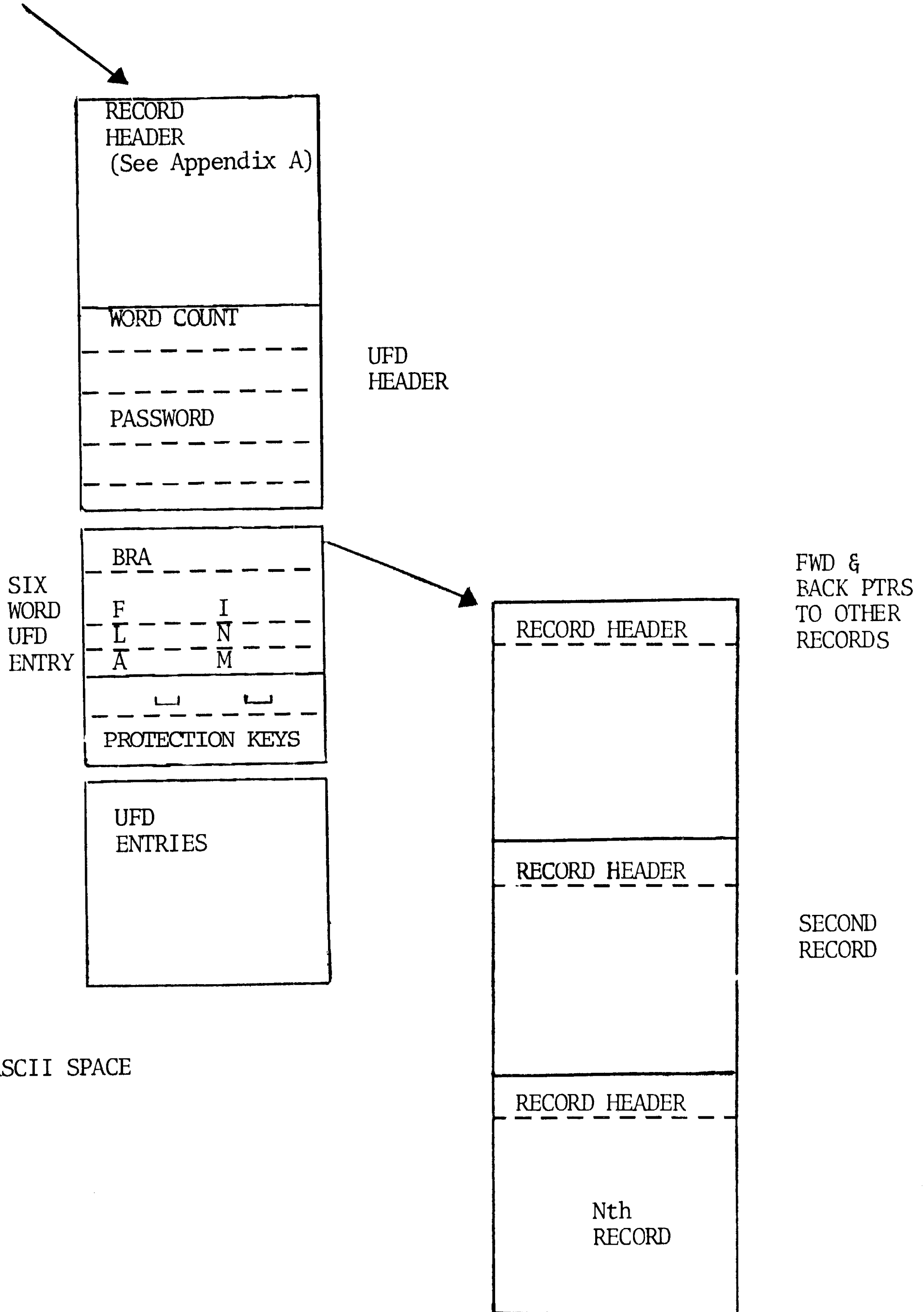


Figure A-1. UFD File Format and Use

Table A-3. Format of DSKRAT

The DSKRAT file has a special header block as follows:

Word	Contents	Meaning
Ø	WRDCNT	Words in header block (5)
1	RECSIZ	Disk record size
2	NRECS	Number of records for file system
3*	CYLS	Cylinder count
4	HEADS	Head count for disk or partition

*not currently used by PRIMOS file system.

WRDCNT allows an expansion of the block size while still maintaining a compatible disk. The header is followed by DSKRAT data, a 1 bit for each record in the file system (NRECS).

During all file transactions, PRIMOS updates the DSKRAT file to reflect the state of records occupied or released, as files or portions of files are added or deleted. The DSKRAT file also contains data on the total disk record count (NRECS).

In PRIMOS III and IV, the name of DSKRAT file(s) may be obtained by using the STATUS command.

Example:

```
OK, STATUS
UFD=ROWDY    Ø

FUNITS
4

DISK    LDEV  PDEV
TSDISK   Ø     51
DUD      1     5Ø
PCBRD    2     52
COMAND   3     54

OK,
```

APPENDIX B

APPLICATION EXAMPLES

GENERAL INFORMATION

This appendix shows a few features of PRIMOS file system software that may help the user in handling data files. The data base structures are defined and then used in several examples that show the speed and flexibility of Prime supporting software.

Features of PRIMOS that facilitate these examples are:

- Sequential File Access (SAM)
- Directed File Access (DAM)
- Segmented structures with multiple growth points
- Relative and absolute positioning
- Pre- and post-access positioning
- Expandable file dimensions
- Security at the UFD, subUFD, and segment directory levels
- Multiple user access to any file (PRIMOS III and IV)
- FORTRAN callable file manager
- Associative buffering (PRIMOS III and IV)
- 2400 RPM moving-head disks
- 30 Megaword storage per device

GENERAL INFORMATION FOR EXAMPLES

Data Base Terminology

To make the discussions and examples that follow meaningful, it is necessary to establish a data base structure. The data base structure consists of three basic structures: data items, data entries, and data sets.

Data Item: The data item is the smallest accessible data element. Each data item is a value and is referenced by a data item name. Usually, many data item values are referenced by the same data item name.

Example:

DATA ITEM NAMES	DATA ITEM VALUES
CITY	DENVER, BOSTON, MIAMI
STATE	COLORADO, MASS, FLORIDA
ZIP	01767, 01752, 07353

The data item is defined as N words (or bytes) of a physical disk record.

Data Entry: The data entry is an ordered collection of related data items and is defined by an ordered listing of the data item names. Data entries are all the same length and are stored in physical disk records.

Example:

DATA ENTRY		DATA ITEM NAMES		
		NAME	CITY	STATE
DISK RECORD	N	SMITH	DENVER	COLORADO
	N+1	JONES	BOSTON	MASS
	N+2	GREEN	MIAMI	FLORIDA

Data Set: The data set is a collection of data entries sharing a common definition. A data set name references any or all of the data entries of a data set. The number of data entries in a data set is limited by available disk space.

There are two types of data sets: master data sets and detail data sets.

Detail Data Set: Detail data sets contain "line item" information, e.g., in the detail data set PERSONNEL, each person's location, education, etc., is stored.

Master Data Set: Master data sets serve as indices to detail data sets. The data entries of a master data set contain pointers to corresponding detail data sets.

In general, access to data within a data base is carried out at the data entry level. Each CALL to an example procedure accesses some or all of the data items within a data entry. The functions provided by the example procedures include adding a new data set, deleting a data entry from a data set, reading some or all of the data items of a data entry, and changing the values of items in a data entry.

Accessing the Data Base

Although access time to specific data in a data base is dependent on the structure of the master and detail data sets, the speed and flexibility of the underlying disk file manager is also significant.

SAM files are a linear array of records threaded with forward and backward pointers. Therefore, to access the last record of a lengthy SAM file (data set) of 47 records, all previous records must be read 47 access times to locate and read the last record. However, the same data set using a DAM file structure would require only three access times to read the same record. The DAM file consists of a record directory maintained by the file system. To access any record in a DAM file takes one disk access to read the directory, and one additional access to read the desired record if it is not the first record in the target file. For 3ØM-word disk, with an average total disk access time of 47.2 ms, the difference is roughly 2-1/2 seconds vs. 1/1Ø of a second.

However, for applications where files are only one or two records in length, the SAM file structure is as fast or faster. DAM files require one disk access just to retrieve the record directory.

Another feature of the SAM file is the way in which the file system computes the best method for locating a record within the file. If the pointer to a SAM file with ten records is positioned at record #8, and the next access addresses record #3, the file system determines whether traversing the file backwards or positioning at the beginning record and traversing forward is faster. In this example, the latter method is selected; thus three accesses instead of five will be made.

Positioning in a SAM or DAM file can be done relatively with a + or - 32767 words (+ or - 74 records) parameter or absolutely with record number, word number parameters. The moving of the file pointer can be pre- or post- the disk access.

For example, let a data set be defined as a SAM or DAM file in a segment directory. A segment directory is a named SAM or DAM file that contains pointers (physical record addresses) to SAM or DAM files (data sets). Thus, each data set has its own growth point. Any number of data sets can be grouped together in a single segment directory providing disk space is available. The segment directory is then defined as the data base.

A tree structure for a file may be developed consisting of a single segment directory and 44Ø DAM files (see Figure B-1). To access a single word out of 85 million (three times the capacity of a 2Ø-surface device) requires only four disk transfers. If repeated accesses are performed over the entire tree, each additional access requires at most three additional disk transfers. If repeated accesses are done within a local part of the file (193,6ØØ words), each additional access requires, at most, one additional disk transfer.

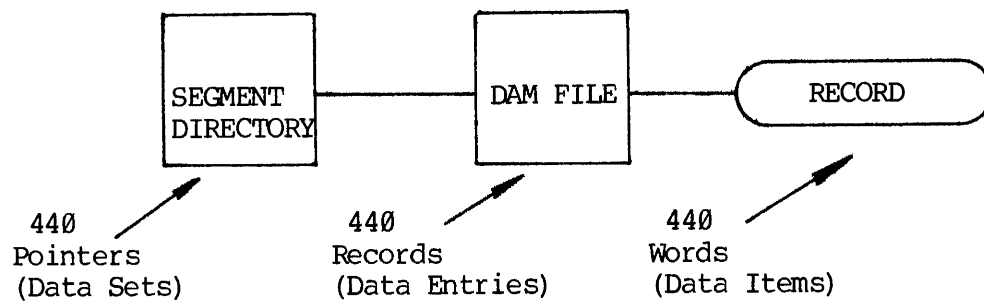


Figure B-1. Sample Tree Structure

In Figure B-1, the segment directory is shown with only 440 pointers, this is not a limit. The segment directory can be expanded just like any ordinary file. However, a segment directory with 440 pointers can directly address 7,744,000 words of data. Segment directories can have "holes". For example, directory entries 1, 3, and 5 can contain valid data set pointers while entries 2 and 4 are not used. This is useful when data sets are arranged logically in a segment directory and additional data sets need to be incorporated later.

A further extension to Prime file structure that may facilitate data base management is the fact that a Keyed Indexed Data access file manager is available to be superimposed upon the PRIMOS III or IV file system. For details of KIDA, refer to the Subroutine Library User Guide (MAN 1880).

FILE SYSTEM PERFORMANCE

No file in the system has a fixed length, providing disk space is available. However, as files become larger, an eventual decrease in performance occurs. For the SAM file, more and more disk accesses are necessary to traverse the file. The DAM file, however, has a boundary where performance falls off. The DAM file directory can handle only 440 record addresses (1024 for a storage module disk); as the file becomes larger, the 441st, 442nd, etc. records are not directly addressable and must be read sequentially. This occurs, however, only when DAM files exceed 193,600 words.

DISK ACCESS TIME

30 million word disk:

	SEEK	ROTATION	ACCESS TIME
Average =	35 ms	12.5 ms	47.5 ms
Maximum =	70 ms	25 ms	95 ms

FILE SECURITY

The PRIMOS III or IV file system has passwords and access attributes associated with the user file directories (UFD´s) and sub-UFD´s. Both the owner and nonowner passwords are defined with the command PASSWD. The PROTECT command allows the association of access attribute with files in a UFD to limit their use if desired. The UFD is always a SAM file and contains up to 72 named files, segment directories, and sub-UFD´s. Once a user is attached to a UFD or sub-UFD, he has access privileges to files in that UFD within limits that may be defined by the PASSWD and PROTECT commands (refer to Man 2602). UFD´s can be

created that contain executable example programs for different levels of security. The user, although attached to the UFD, has no way to dump, modify, or delete the executable programs if the command directory (CMDNC0) was empty or protected by an owner password unknown to him, or by a combination of passwords and protection attributes.

Under multi-user (PRIMOS III or IV), more than one user has access to a file simultaneously, provided it is opened for reading only. If on the other hand, the file is opened for writing by one user, other users are prevented through a locking algorithm from reading or writing it.

Under PRIMOS III and IV, associative buffering is implemented with 32 buffers. Each buffer contains one disk record (440 words). A Least Recently Used (LRU) algorithm is used when a record not in the buffer is accessed. This greatly decreases access time because the data set directories (the indexes to all data) tend to remain in buffers because they are frequently referenced.

To further illustrate the capability of the file system, several examples are given that show different data base structures, and the access times and resources required when traversing them.

Example #1

This example uses a file structure consisting of a segment directory with 44Ø SAM files and then with 44Ø DAM files to show the difference in average access time. All the files are five records long, (22ØØ words).

44Ø x 22ØØ (968,ØØØ words)

SAM FILES			DAM FILES		
Seg. Dir	44Ø	(SAM)	Seg. Dir.	44Ø	(SAM)
	[1]		DAM Dir.	[1]	
	[2]	SAM FILE		[2]	DAM FILE
	[3]	(22ØØ Words)		[3]	(22ØØ Words)
	[4]			[4]	
	[5]			[5]	

	DISK ACCESSES		DISK ACCESSES
OPEN Seg. Dir.	1	OPEN Seg. Dir.	1
OPEN SAM file	1	OPEN DAM file	1
AVG file access	2	AVG file access	1
TOTAL	4	TOTAL	3

Example #2

This example is similar to Example #1 but with much larger files. All files are 50 records long (22,000 words).

440 x 22,000 (9,680,000 words)

	DISK ACCESSES		DISK ACCESSES
OPEN Seg. Dir.	1	OPEN Seg. Dir.	1
OPEN SAM file	1	OPEN DAM file	1
AVG file access	24.5	AVG file access	1
TOTAL	26.5	TOTAL	3

Example #3

This example uses two levels of segment directories and the SAM/DAM comparison. The first directory contains pointers to ten other segment directories which each contain pointers to 44Ø files. The files are five records long (22ØØ words).

1Ø x 44Ø x 22ØØ (9,68Ø,ØØØ words)

SAM FILES				DAM FILES	
1st Seg. Dir.	1Ø	(SAM)		1st Seg. Dir.1Ø(SAM)	
2nd Seg. Dir.	44Ø	(SAM)		2nd Seg. Dir.44Ø(SAM)	
[1]		DAM Dir.		[1]	
[2]				[2]	DAM FILE
[3]	SAM FILE			[3]	(22ØØ words)
[4]	(22ØØ words)			[4]	
[5]				[5]	

DISK ACCESSES		DISK ACCESS	
OPEN Seg. Dir. #1	1	OPEN Seg. Dir. #1	1
OPEN Seg. Dir. #2	1	OPEN Seg. Dir. #2	1
OPEN SAM file	1	OPEN DAM file	1
AVG file access	2	AVG file access	1
	-		-
TOTAL	5	TOTAL	4

Example #4

Example #4 shows a more complicated file structure: a segment directory that contains one pointer to a master index file (DAM), 5Ø pointers to master data sets (DAM files), and 5Ø pointers to data segment directories. The data segment directories contain the pointers to the detail data sets (SAM files of 88Ø words).

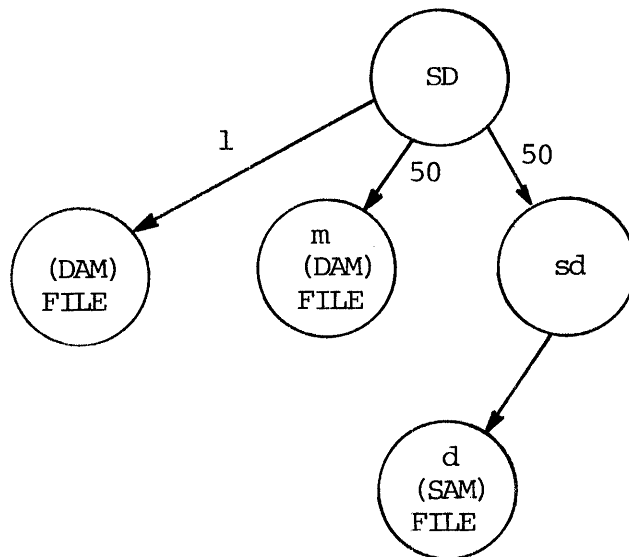


Figure B-2 Sample File Structure

The time and resources to read randomly into the file structure are shown in Figure B-2 are as follows:

Initialize	DISK ACCESSES	BUFFERS	UNITS
OPEN SD	1	1	1
OPEN sd	1	1	1
OPEN M	2	2	1
	4		

Read master directory to select index

ACCESS M	1	Ø	Ø

Read index

OPEN m	2	2	1

ACCESS m	1	Ø	Ø
Read data			
OPEN SAM	1	1	1
ACCESS SAM	.5	Ø	Ø
	5.5	7	5

The maximum number of buffers available under PRIMOS II is 16, under PRIMOS III there are 32. Example #4 shows seven buffers open which occupies 3.1K words of memory. There are 16 units available under PRIMOS.

APPENDIX C

USE OF PRIMOS FILE SYSTEM

INTRODUCTION

This appendix gives guidance in and examples of how to use the file system. The expanded key definitions of SEARCH, PRWFIL, and ATTACH have been rewritten in this appendix with mnemonic keys. The following examples use variables defined and initialized by the insert file KEYCOM.

A user wishing to use these keys must have the statement INSERT KEYCOM in his FORTRAN program after the storage specification statements and before any data segments. The user will have to copy KEYCOM to the appropriate UFD before compiling the program(s). This appendix provides examples of use of the file system (refer to "Examples".)

The following example programs are:

Program Name	Function
KEYCOM	Provides mnemonic keys for PRWFIL SEARCH, and ATTACH.
SAMWRT	Writes a SAM data file.
DAMWRT	Writes a DAM data file.
REDFIL	Reads a SAM or DAM file of unlimited length and prints the largest integer in the file. This program also shows how to use alternate return.
RDLREC	Reads logical record number n from a file of fixed-length recorded.
CRTSEG	Creates a segment directory.
REDSEG	Reads file on a segment directory and prints a specified word (record) in that file.
RDVREC	Reads logical record number n from a file on variable-length records.
GPTRFL	Generates a pointer file that consists of two-word pointers to each logical record in another file.

```
C KEYCOM  JPC 30 MAY 1974
C  PROVIDES MNEMONIC KEYS FOR PRWFIL, SEARCH, AND ATTACH
      INTEGER PREAD,PWRITE,PREREL,PREABS,POSREL,POSABS,PCONV,
     X    OPNRED,OPNWRT,OPNBTH,CLOSE,DELETE,REWIND,
     X    TRNCAT,UFDREF,SEGREF,NTFILE,NDFILE,NTSEG,NDSEG,NEWUFD,
     X    MFDUFD,CURUFD,SEGUFD,HOMUFD,SETHOM
C
      DATA PREAD,PWRITE,PREREL,PREABS,POSREL,POSABS,PCONV
     X   / :1,    :2,     :0,     :10,    :20 ,  :30,    :400/
      DATA OPNRED,OPNWRT,OPNBTH,CLOSE,DELETE,REWIND,TRNCAT
     X   / 1,     2,      3,      4,     5,      7,      8  /
      DATA UFDREF,SEGREF,NTFILE,NDFILE,NTSEG,NDSEG,NEWUFD
     X    /:0,    :100,  :0,    :2000, :4000,:6000,:10000/
      DATA MFDUFD,CURUFD,SETHOM
     X   / 0,     2,      1    /
```

```
C SAMWRT, CARLSON JULY 10, 1974
C
C PROGRAM SAM-WRITE TO WRITE A SAM DATA FILE
C
C THE FILE IS 1000 WORDS WRITTEN FROM ARRAY BUFF.
C
C RESTRICTIONS: SAMFIL SHOULD NOT EXIST BEFORE RUNNING THE PROGRAM.
C
C
      INTEGER BUFF(1000),PBUFF,FUNIT1
C
C VARIABLE DEFINITIONS:
C BUFF- ARRAY TO BE WRITTEN TO A FILE
C PBUFF- POINTER TO BUFF
C FUNIT1- CONTAINS 1, REFERS TO FILE UNIT 1
C
C ROUTINES CALLED
C LOC,SEARCH,PRWFIL,EXIT
C
C
C KEYCOM CONTAINS FILE KEY DEFINITIONS
C
$INSERT KEYCOM
C
C
      DATA FUNIT1/1/
C
C
C INITIALIZE BUFFER CONTENTS
      DO 10 I=1,1000
        BUFF(I)=I
10    CONTINUE
C
C LOC RETURNS A POINTER TO ITS ARGUMENT
C
      PBUFF=LOC(BUFF)
C
C OPEN A NEW SAM DATA FILE CALLED SAMFIL IN THE CURRENT UFD
C FOR WRITING ON FILE UNIT 1.
C ON MOST CALLS THE UFDREF KEY IS OMITTED SINCE ITS VALUE IS 0.
C
C THE FOLLOWING STATEMENT WILL BE COMPILED AS IF IT WERE WRITTEN
C TEMP=OPNWRT+NTFILE+UFDREF
C CALL SEARCH(TEMP,'SAMFIL',FUNIT1,0)
C THE USE OF MULTIPLE MNEMONIC KEYS WILL GENERATE MORE CODE THAN
C THE USE OF NUMERIC KEYS.
C
      CALL SEARCH(OPNWRT+NTFILE+UFDREF,'SAMFIL',FUNIT1,0)
C
C WRITE 1000 WORDS FROM BUFF INTO FILE UNIT 1.
C
      CALL PRWFIL(PWRITE,FUNIT1,PBUFF,1000,0,0)
C
```

```
C CLOSE FILE. THIS RELEASES FILE UNIT 1 FOR REUSE AND INSURES
C ALL FILE BUFFERS HAVE BEEN WRITTEN TO THE DISK.
C
      CALL SEARCH(CLOSE,0,FUNIT1,0)
C
C RETURN TO PRIMOS
C
      CALL EXIT
C
      END
```

```
C REDFIL, CARLSON, JULY 10,1974
C
C PROGRAM READ-FILE TO READ A SAM OR DAM FILE OF UNLIMITED LENGTH
C AND PRINT THE LARGEST INTEGER IN THE FILE.
C
C THIS PROGRAM SHOWS HOW TO USE THE ALTERNATE RETURN FEATURE
C OF SEARCH AND PRWFIL AND HOW TO USE GETERR AND PRERR IN
C CONJUNCTION WITH THE ALTERNATE RETURN. NOTE THAT THE PROGRAM
C DOESN'T CHECK IF THE FILE IS SAM OR DAM. TO THE USER, SAM AND
C DAM FILES ARE FUNCTIONALLY EQUIVALENT.
C
C RESTRICTIONS: NONE
C
C
      INTEGER BUFF(100),PBUFF,UERVEC(4),FUNIT1,LARGST,FNAME(3),N
C
C VARIABLE DEFINITIONS
C BUFF- BUFFER TO HOLD INFORMATION READ FROM FILE
C PBUFF- POINTER TO BUFF
C UERVEC- USER ERROR VECTOR. HOLDS ERROR VECTOR OBTAINED FROM PRIMOS
C FUNIT1- CONTAINS 1, USED TO REFER TO FILE UNIT 1
C LARGST- VARIABLE TO HOLD LARGEST INTEGER IN FILE
C FNAME- HOLDS A FILE NAME
C
C ROUTINES CALLED
C LOC,SEARCH,PRWFIL,GETERR,PRERR,EXIT
C
$INSERT KEYCOM
C
      DATA FUNIT1/1/
C
C
C INITIALIZATION
C
      PBUFF=LOC(BUFF)
      LARGST=-32767
C
C ASK USER FOR FILE NAME. FORTRAN UNIT 1 IS THE USER TERMINAL.
C
10    WRITE(1,1000)
1000  FORMAT('TYPE FILE NAME')
C
C READ FILE NAME
      READ(1,1010)(FNAME(I),I=1,3)
1010  FORMAT(3A2)
C
C OPEN FNAME IN THE CURRENT UFD FOR READING ON FILE UNIT 1.
C IF ANY ERROR, GO TO LABEL 100.
C
      CALL SEARCH(OPNRED,FNAME,FUNIT1,$100)
C
C READ FILE 100 WORDS AT A TIME. SET LARGST TO THE LARGEST INTEGER
C READ. WHEN END OF FILE IS REACHED, THE ALTERNATE RETURN OF
```

```
C PRWFIL SENDS CONTROL TO LABEL 50.
C
30     CALL PRWFIL(PREAD,FUNIT1,PBUFF,100,0,$50)
C
C 100 WORDS READ INTO BUFF, SET LARGST
C
       DO 40 I=1,100
         IF (LARGST.LE.0.AND.BUFF(I).GE.0) LARGST=BUFF(I)
C
C THE ABOVE TEST IS DONE BECAUSE IF BUFF(I)-LARGST IS GREATER
C THAN 32767, THE FOLLOWING COMPARISON FAILS DUE TO ARITHMETIC OVERFLOW
C
         IF (LARGST.LT.BUFF(I)) LARGST=BUFF(I)
40     CONTINUE
C
C LOOP BACK TO READ MORE DATA FROM FILE
C
       GO TO 30
C
C ALTERNATE RETURN TAKEN ON PRWFIL. SET ERROR TYPE FROM ERRVEC
C THROUGH A CALL TO GETERR.
C
50     CALL GETERR(UERVEC,4)
C
C IF ERROR TYPE NOT END OF FILE (CODE 'PE'), PRINT THE
C ERROR MESSAGE WITH PRERR AND RETURN TO PRIMOS.
C
       IF (UERVEC(1).EQ.'PE') GO TO 60
         CALL PRERR
         CALL EXIT
C
C END OF FILE. NUMBER OF WORDS IN PRWFIL CALL LEFT TO BE
C TRANSFERRED IS IN UERVEC(2)
C N IS SET TO NUMBER OF WORDS TRANSFERRED ON LAST CALL.
C
60     N=100-UERVEC(2)
       IF (N.EQ.0) GO TO 80
C
C SET LARGST
C
       DO 70 I=1,N
         IF (LARGST.LE.0.AND.BUFF(I).GE.0) LARGST=BUFF(I)
         IF (LARGST.LT.BUFF(I)) LARGST=BUFF(I)
70     CONTINUE
C
C THE FOLLOWING PRWFIL CALL ACTS AS A NO-OPERATION ON THE FILE
C BUT PUTS THE FILE POINTER IN ERRVEC.
C
80     CALL PRWFIL(PREAD,FUNIT1,0,0,0,0)
       CALL GETERR(UERVEC,4)
C
C FILE POINTER IS (V-RECORD-NO.,WORD-NO.) IN UERVEC(3) AND UERVEC(4).
C IF FILE POINTER IS (0,0) AT THIS POINT, IT INDICATES THE FILE
```

```
C CONTAINS NO DATA.
C
      IF (UERVEC(3).EQ.Ø.AND.UERVEC(4).EQ.Ø) GO TO 11Ø
C
C FILE NOT EMPTY, PRINT LARGST.
C
      WRITE(1,1Ø2Ø)LARGST
1Ø2Ø  FORMAT('LARGEST INTEGER IN FILE IS 'I6)
C
C CLOSE FILE AND RETURN TO PRIMOS
C
9Ø    CALL SEARCH(CLOSE,Ø,FUNIT1,Ø)
      CALL EXIT
C
C ERROR IN ATTEMPT TO OPEN FILE
C PRINT MESSAGE AND GET ERROR CODE.
C
1ØØ   CALL PRERR
      CALL GETERR(UERVEC,1)
C
C IF ERROR IS NAME NOT FOUND (CODE 'SH'), GO TO LABEL 1Ø TO ASK
C FOR A NEW NAME OTHERWISE GIVE UP AND RETURN TO PRIMOS.
C
      IF (UERVEC(1).EQ.'SH') GO TO 1Ø
      CALL EXIT
C
C FILE EMPTY
C
11Ø   WRITE(1,1Ø3Ø)
1Ø3Ø  FORMAT('FILE EMPTY')
      GO TO 9Ø
C
      END
```

```
C RDLREC, CARLSON, JULY 10,1974
C
C RDLREC- READ LOGICAL RECORD
C PROGRAM TO READ LOGICAL RECORD NUMBER N FROM A FILE CONSISTING
C OF FIXED LENGTH RECORDS
C
C IN THIS PROGRAM THE FILE ACCESSED IS CONSIDERED TO CONTAIN
C AN UNLIMITED NUMBER OF LOGICAL RECORDS, EACH RECORD CONSISTING
C OF M WORDS. THE PROGRAM READS AND TYPES THE CONTENTS OF RECORD
C NUMBER N AS M INTEGERS.THE FIRST RECORD OF A FILE IS RECORD NUMBER 0.
C NOTE THAT A LOGICAL RECORD IS MEARLY A GROUPING OR WORDS IN
C A FILE. IT HAS NO RELATION TO THE PHYSICAL DISK RECORD.
C
C RESTRICTIONS: RECORD SIZE MUST BE BETWEEN 1 AND 1000
C               RECORD NUMBER MUST BE BETWEEN 0 AND 32767
C               (RECORD-SIZE)*(RECORD-NUMBER) MUST BE LESS THAN
C               8,388,608 (2**23) FLOATING POINT NUMBERS-ONLY REPRESENT
C               6.8 DIGITS.
C               THE RECORD MUST BE IN THE FILE
C
C
      INTEGER PBUFF,BUFF(1000),FUNIT1,FNAME(3),RECSIZ,RECNUM,POSITN,
     X   ABSPOS(2)
C
      REAL FRECSZ,FRCNUM,FPOSTN,PRECSZ
C
C VARIABLE DEFINTIONS
C BUFF- BUFFER USED TO HOLD A LOGICAL RECORD
C PBUFF- POINTER TO BUFFER
C FUNIT1- CONTAINS 1, USED TO REFER TO FILE UNIT 1
C FNAME- HOLDS A FILE NAME
C RECSIZ- LOGICAL RECORD SIZE
C RECNUM- LOGICAL RECORD NUMBER
C POSITN- RELATIVE POSITION TO POSITION TO REQUESTED RECORD
C ABSPOS- ABSOLUTE POSITION TO POSITON TO REQUESTED RECORD
C FRCNUM- FLOATING POINT LOGICAL RECORD NUMBER
C FRECSZ- FLOATING POINT LOGICAL RECORD SIZE
C FPOSTN- FLOATING POINT POSITION NEEDED TO POSITION TO REQUESTED RECOR
D
C VRECSZ- V-RECORD DATA SIZE. USED TO FORM
C         TWO WORD ABSOLUTE POSITION.
C
C ROUTINES CALLED
C LOC,SEARCH,FLOAT,INT,AMOD,GINFO,PRWFIL,EXIT,GETERR,PRERR
C
$INSERT KEYCOM
C
      DATA FUNIT1/1/
      DATA VRECSZ/440/
C
C INITIALIZATION
C
      PBUFF=LOC(BUFF)
```

```
C
C ASK FOR FILE NAME
C
      WRITE(1,1000)
1000  FORMAT('TYPE FILE NAME')
C
C READ FILE NAME
C
      READ'1,1010)(FNAME(I),I=1,3)
1010  FORMAT(3A2)
C
C OPEN FNAME IN THE CURRENT UFD FOR READING ON FILE UNIT 1
C
      CALL SEARCH(OPNRED,FNAME,FUNIT1,0)
C
C ASK FOR RECORD SIZE
C
20    WRITE(1,1020)
1020  FORMAT('TYPE RECORD SIZE')
      READ(1,1030)RECSIZ
1030  FORMAT(I6)
      IF (RECSIZ.GE.1.AND.RECSIZ.LE.1000) GO TO 30
         WRITE(1,1040)
1040  FORMAT('BAD RECORD SIZE')
         GO TO 20
C
C ASK FOR RECORD NUMBER. FIRST RECORD IS NUMBERED 0.
C
30    WRITE(1,1050)
1050  FORMAT('TYPE RECORD NUMBER')
      READ(1,1030) RECNUM
      IF (RECNUM.GE.0) GO TO 35
      WRITE(1,1051)
1051    FORMAT('BAD RECORD NUMBER')
      GO TO 30
C
C CHECK IF RECORD IS MORE THAN 32767 WORDS FROM BEGINNING OF
C FILE. IF SO, USE ABSOLUTE POSITIONING ELSE USE RELATIVE
C POSITIONING.
C
35    FRECSZ=FLOAT(RECSIZ)
      FRCNUM=FLOAT(RECNUM)
      FPOSTN=FRECSZ*FRCNUM
      IF (FPOSTN.LT.8388608.) GO TO 40
        WRITE(1,1055)
1055    FORMAT('RECORD-NUMBER*RECORD-SIZE IS TOO LARGE')
        GO TO 20
40    IF (FPOSTN.GT.32767.) GO TO 100
C
C RECORD IS LESS THAN 32767 WORDS FROM BEGINNING, USE RELATIVE
C POSITIONING.
C NOTE THAT ABSOLUTE POSITIONING COULD HAVE BEEN USED FOR A RECORD
C ANYWHERE IN THE FILE, NOT JUST FOR THOSE RECORDS BEYOND WORD
```

```
C 32767. RELATIVE IS SHOWN IN HERE ONLY FOR AN EXAMPLE.
C
      POSITN=RECSIZ*RECNUM
C
C POSITION TO THE RECORD AND READ RECSIZ WORDS INTO THE BUFFER
C
      CALL PRWFIL(PREAD+PREREL,FUNIT1,PBUFF,RECSIZ,POSITN,$300)
C
C GO TO 200 TO TYPE RECORD CONTENTS
C
      GO TO 200
C
C RECORD MORE THAN 32767 WORDS FROM BEGINNING OF FILE, USE
C ABSOLUTE POSITIONING
C
C CALCULATE ABSOLUTE POSTION (V-RECORD-NUMBER,WORD-NUMBER)
C THAT RECORD STARTS AT AND PUT IN ABSPOS(1) AND ABSPOS(2)
C
      ABSPOS(1)=INT(FPOSTN/VRECSZ)
      ABSPOS(2)=INT(AMOD(FPOSTN,VRECSZ))
C
C POSITION TO THE RECORD AND READ RECSIZ WORDS INTO THE BUFFER
C
      CALL PRWFIL(PREAD+PREABS,FUNIT1,PBUFF,RECSIZ,ABSPOS,$300)
C
C RECORD READ, NOW TYPE IT.
C
200   WRITE(1,1060) RECNUM,RECSIZ
1060  FORMAT('RECORD 'I6,' CONTAINS 'I6,' ENTRIES AS FOLLOWS')
      WRITE(1,1070)(BUFF(I),I=1,RECSIZ)
1070  FORMAT(10I7)
C
C RETURN TO PRIMOS AFTER CLOSING THE FILE
C
      CALL SEARCH(CLOSE,0,FUNIT1)
      CALL EXIT
C
C ERROR WHILE ATTEMPTING TO READ THE RECORD
C
300   CALL GETERR(BUFF,1)
      CALL PRERR
      IF (BUFF(1).EQ.'PE') GO TO 305
      CALL EXIT
C
C END OF FILE REACHED, REWIND FILE AND TRY AGAIN.
C
305   CALL SEARCH(REWIND,0,FUNIT1,0)
      GO TO 20
C
      END
```

```
C CRTSEG, CARLSON, JULY 12, 1974
C
C CRTSEG- CREATE-SEGMENT-DIRECTORY
C THIS PROGRAM SHOWS HOW TO CREATE A SEGMENT DIRECTORY AND WRITE
C FILES INTO IT.
C
C RESTRICTIONS: SEGDIR SHOULD NOT EXIST BEFORE RUNNING THE PROGRAM.
C
      INTEGER PBUFF,BUFF(1Ø),SGUNIT,FUNIT
C
C
C VARIABLE DEFINITIONS
C BUFF- BUFFER TO WRITE TO SEGMENT DIRECTORY FILES
C PBUFF- POINTER TO BUFF
C SGUNIT- CONTAINS 1, FILE UNIT USED FOR SEGMENT DIRECTORY
C FUNIT- CONTAINS 2, FILE UNIT USED FOR DATA FILES
C
$INSERT KEYCOM
C
      DATA SGUNIT,FUNIT/1,2/
C
C
C INITIALIZATION
C
      PBUFF=LOC(BUFF)
      DO 1Ø I=1,1Ø
        BUFF(I)=I
1Ø    CONTINUE
C
C OPEN A NEW SAM SEGMENT DIRECTORY CALLED SAMDIR IN THE
C CURRENT UFD FOR READING AND WRITING ON FILE UNIT SGUNIT.
C
      CALL SEARCH(OPNBTH+NTSEG+UFDREF,'SEGDIR',SGUNIT,Ø)
C
C OPEN A NEW SAM DATA FILE FOR WRITING ON FILE UNIT FUNIT. WRITE
C THE DISK LOCATION OF THIS NEW FILE AT THE FILE POINTER OF
C THE SEGMENT DIRECTORY OPEN ON FILE UNIT SGUNIT.
C THE FILE POINTER POINTS TO WORD NUMBER Ø OF THE SEGMENT DIRECTORY.
C
      CALL SEARCH(OPNWRT+NTFILE+SEGREF,SGUNIT,FUNIT,Ø)
C
C WRITE 1Ø WORDS FROM BUFF INTO THE DATA FILE
C
      CALL PRWFIL(PWRITE,FUNIT,PBUFF,1Ø,Ø,Ø)
C
C CLOSE THE DATA FILE
C
      CALL SEARCH(CLOSE,Ø,FUNIT,Ø)
C
C REPLACE BUFF WITH NEW DATA
C
      DO 2Ø I=1,1Ø
        BUFF(I)=I*1Ø
```

```
20    CONTINUE
C
C OPEN A DIFFERENT NEW SAM DATA FILE ON FUNIT. PUT THE
C DISK LOCATION IN WORD NUMBER 1 OF THE SEGMENT DIRECTORY.
C THIS IS DONE IN TWO STEPS: FIRST, BY POSITIONING THE FILE
C POINTER OF THE SEGMENT DIRECTORY FORWORD ONE WORD, AND THEN
C BY CALLING SEARCH AS SHOWN ABOVE.
C
      CALL PRWFIL(PREAD+PREREL,SGUNIT,0,1,0)
      CALL SEARCH(OPNWRT+NTFILE+SEGREF,SGUNIT,FUNIT,0)
C
C WRITE 10 WORDS IN THE FILE
C
      CALL PRWFIL(PWRITE,FUNIT,PBUFF,10,0,0)
C
C CLOSE THE DATA FILE
C
      CALL SEARCH(CLOSE,0,FUNIT,0)
C
C CLOSE THE SEGMENT DIRECTORY
C
      CALL SEARCH(CLOSE,0,SGUNIT,0)
C
C RETURN TO PRIMOS
C
      CALL EXIT
C
      END
```

```
C REDSEG, CARLSON, JULY 12, 1974
C
C REDSEG- READ-FILE-IN-SEGMENT-DIRECTORY
C THIS PROGRAM READS FILE NUMBER N IN A SEGMENT DIRECTORY AND
C TYPES WORD NUMBER M IN THAT FILE. THE FIRST FILE IN THE DIRECTORY
C IS FILE NUMBER Ø. THE FIRST WORD IN THE FILE IS WORD NUMBER Ø.
C
C RESTRICTIONS: THE FILE NUMBER MUST BE BETWEEN Ø AND 32767.
C               THE FILE MUST BE IN THE SEGMENT DIRECTORY.
C               THE WORD NUMBER MUST BE BETWEEN Ø AND 32767.
C               THE WORD MUST BE IN THE FILE.
C
      INTEGER PBUFF,BUFF,SGUNIT,FUNIT,SEGDIR(3),UERVEC(2),FILNUM,
     X   WRDNUM
C
C VARIABLE DEFINITIONS
C BUFF- HOLDS WRDNUM WORD OF FILNUM FILE OF SEGDIR
C PBUFF- POINTER TO BUFF
C SGUNIT- CONTAINS 1, FILE UNIT USED FOR SEGMENT DIRECTORY
C FUNIT- CONTAINS 2, FILE UNIT USED FOR DATA FILE
C UERVEC- HOLDS ERROR VECTOR OBTAINED FROM DOS
C FILNUM- HOLDS FILE NUMBER OF SEGDIR TO READ
C WRDNUM- HOLDS WORD NUMBER OF NTH FILE TO READ
C SEGDIR- HOLDS SEGMENT DIRECTORY NAME
C
$INSERT KEYCOM
C
      DATA SGUNIT,FUNIT/1,2/
C
C
C INITIALIZATION
C
      PBUFF=LOC(BUFF)
C
C ENSURE UNITS ARE CLOSED AND
C ASK FOR AND READ SEGMENT DIRECTORY NAME
C
1Ø    CALL SEARCH(CLOSE,Ø,SGUNIT,Ø)
      CALL SEARCH(CLOSE,Ø,FUNIT,Ø)
      WRITE(1,1ØØØ)
1ØØØ  FORMAT('TYPE SEGMENT DIRECTORY NAME')
      READ(1,1Ø1Ø)(SEGDIR(I),I=1,3)
1Ø1Ø  FORMAT(3A2)
C
C OPEN THE SEGMENT DIRECTORY FOR READING ON SGUNIT
C
      CALL SEARCH(OPNRED+UFDREF,SEGDIR,SGUNIT,Ø)
C
C GET FILE TYPE FROM ERRVEC AND MAKE SURE FILE IS A SEGMENT DIRECTORY.
C ALLOWABLE TYPE CODES ARE SAMSEG AND DAMSEG., VALUES 2 AND 3.
C
      CALL GETERR(UERVEC,2)
      IF (UERVEC(2).EQ.2.OR.UERVEC(2).EQ.3) GO TO 2Ø
```

```
C
C NOT A SEGMENT DIRECTORY, TRY AGAIN.
C
      WRITE(1,1020)
1020  FORMAT('FILE IS NOT A SEGMENT DIRECTORY')
      GO TO 10
C
C ASK FOR FILE IN SEGMENT DIRECTORY
C
20    WRITE(1,1030)
1030  FORMAT('TYPE FILE NUMBER')
      READ(1,1040) FILNUM
1040  FORMAT(I6)
C
C ASK FOR  WORD IN FILE
C
      WRITE(1,1035)
1035  FORMAT('TYPE WORD NUMBER')
      READ(1,1040) WRDNUM
C
C TRY TO POSITION TO FILNUM FILE IN THE SEGMENT DIRECTORY.
C IF ERROR GO TO 100
C
      CALL PRWFIL(PREAD+PREREL,SGUNIT,0,0,FILNUM,$100)
C
C OPEN FILE IN SEGMENT DIRECTORY FOR READING ON FUNIT
C GO TO 120 IF ANY ERROR.
C
      CALL SEARCH(OPNRED+SEGREF,SGUNIT,FUNIT,$120)
C
C POSITION TO FILWRD WORD IN DATA FILE AND READ IT INTO BUFF
C GO TO 200 IF ANY ERROR.
C
      CALL PRWFIL(PREAD+PREREL,FUNIT,PBUFF,1,WRDNUM,$200)
C
C PRINT THE WORD, CLOSE FILES AND RETURN TO PRIMOS
C
      WRITE(1,1050)WRDNUM,FILNUM,(SEGDIR(I),I=1,3),BUFF
1050         FORMAT('WORD'I6,' OF FILE'I6,' IN '3A2,' CONTAINS'I6)
50    CALL SEARCH(CLOSE,0,FUNIT,0)
      CALL SEARCH(CLOSE,0,SGUNIT,0)
      CALL EXIT
C
C FILE NOT IN SEGMENT DIRECTORY
C 'PE' IS THE CODE FOR PRWFIL EOF
C 'PG' IS THE CODE FOR PRWFIL BOF
C
100   CALL GETERR(UERVEC,1)
      IF (UERVEC(1).EQ.'PE'.OR.UERVEC(1).EQ.'PG') GO TO 110
        CALL PRERR
        GO TO 50
C
110   WRITE(1,1060)(SEGDIR(I),I=1,3)
```

```
1060  FORMAT('FILE NOT IN '3A2)
      GO TO 10
C
C ERROR IN ATTEMPTING TO OPEN FILE IN SEGMENT DIRECTORY
C
120   CALL GETERR(UERVEC,1)
C
C SEE IF SEGMENT DIRECTORY ERROR TYPE
C
      IF (UERVEC(1).EQ.'SQ') GO TO 130
        CALL PRERR
        CALL EXIT
C
C YES, FILE POINTER IF SGUNIT IS AT END OF FILE OR DISK ADDRESS
C OF FILE IS 0 INDICATING NO FILE AT THIS FILE POINTER.
C IN EITHER CASE, THE ERROR INDICATES THE REQUESTED FILE IS NOT
C IN THE SEGMENT DIRECTORY.
C THIS ERROR CODE IS ALSO GIVEN IF NO FILE IS OPEN FOR READING
C ON SGUNIT IF SEARCH IS OPENING AN EXISTING FILE IN A SEGMENT
C DIRECTORY OR IF NO FILE IS OPEN FOR BOTH READING AND WRITING
C ON SGUNIT IF SEARCH IS OPENING A NEW FILE IN A SEGMENT DIRECTORY.
C
C THESE ERROR CONDITIONS CAN NEVER OCCUR IN THIS PROGRAM.
C
130   GO TO 110
C
C WORD NOT IN FILE
C
200   CALL GETERR(UERVEC,1)
      IF (UERVEC(1).EQ.'PE'.OR.UERVEC(1).EQ.'PG') GO TO 210
        CALL PRERR
      CALL EXIT
C
210   WRITE(1,1070)WRDNUM,FILNUM,(SEGDIR(I),I=1,3)
1070  FORMAT('WORD'I6,' NOT IN FILE'I6,' IN '3A2)
      GO TO 10
C
      END
```

```
C RDVREC, CARLSON, JULY 16,1974
C
C RDVREC- READ-VARIABLE-LENGTH-RECORD
C PROGRAM TO READ LOGICAL RECORD NUMBER N FROM A FILE CONSISTING
C OF A GROUP OF VARIABLE LENGTH RECORDS AND TYPE THE RECORD
C ON THE TERMINAL.
C
C THE FILE VARREC CONSISTS OF LOGICAL RECORDS. EACH LOGICAL
C RECORD CONSISTS OF A HEADER WORD WHICH CONTAINS THE SIZE
C OF THE RECORD FOLLOWED BY THE DATA IN THE RECORD.
C THE FIRST RECORD OF THE FILE IS RECORD NUMBER  0.
C
C THE METHOD USED IS: FIRST TO GENERATE PTRFIL, AN
C ANCILLARY FILE OF 2 WORD POSITION POINTERS TO EACH RECORD
C IN THE FILE VARREC. THIS IS DONE BY THE PROGRAM GPTRFL
C (GENERATE POINTER FILE) FOLLOWING THIS PROGRAM. RDVREC
C USES THE NTH FILE POINTER IN PTRFIL TO ACCESS THE NTH LOGICAL
C RECORD IN VARREC. NOTE THAT PTRFIL NEEDS TO BE GENERATED
C ONLY ONCE. AFTER THAT THE USER CAN MAKE ANY NUMBER OF
C REFERENCES TO VARREC. FOR FAST ACCESS, BOTH PTRFIL
C AND VARREC SHOULD BE GENERATED AS DAM FILES. HANDLING OF PRWFIL
C ERRORS IS OMITTED TO SIMPLIFY THIS EXAMPLE.
C
C RESTRICTIONS: FILE VARREC MUST EXIST IN THE CURRENT UFD.
C               FILE PTRFIL MUST EXIST IN THE CURRENT UFD.
C               RECORD SIZE MUST BE BETWEEN 1 AND 1000.
C               THE RECORD REQUESTED MUST BE BETWEEN 0 AND 16383.
C               THE RECORD MUST BE IN THE FILE VARREC.
C
      INTEGER FUNIT,SIZE,RECNUM,ABSPOS(2),PABSPS,BUFF(1000),
     X   PBUFF,PBUFF2
C
C VARIABLE DEFINITIONS
C FUNIT- CONTAINS 1, USED TO REFER TO FILE UNIT 1
C SIZE- HOLDS SIZE OF LOGICAL RECORD
C RECNUM- HOLDS LOGICAL RECORD NUMBER REQUESTED
C ABSPOS- HOLDS FILE POINTER
C PABSPS- POINTER TO ABSPOS
C BUFF- HOLDS RECNUM LOGICAL RECORD
C PBUFF- POINTER TO BUFF
C PBUFF2- POINTER TO BUFF(2)
C
C ROUTINES CALLED
C SEARCH,PRWFIL,EXIT,LOC
C
$INSERT KEYCOM
C
      DATA FUNIT/1/
C
C
C INITIALIZATION
C
      PABSPS=LOC(ABSPOS)
```

```
      PBUFF=LOC(BUFF)
      PBUFF2=LOC(BUFF(2))
C
C ASK FOR RECORD NUMBER. FIRST RECORD IS NUMBERED Ø.
C
      WRITE(1,1ØØØ)
1ØØØ  FORMAT('TYPE RECORD NUMBER')
      READ(1,1Ø1Ø) RECNUM
1Ø1Ø  FORMAT(I6)
C
C OPEN FILE OF 2-WORD FILE POINTERS CALLED PTRFIL ON FUNIT
C
      CALL SEARCH(OPNRED,'PTRFIL',FUNIT,Ø)
C
C POSITION TO REQUESTED FILE POINTER AND READ IT INTO ABSPOS
C
      CALL PRWFIL(PREAD+PREREL,FUNIT,PABSPS,2,RECNUM*2,Ø)
C
C CLOSE FUNIT
C
      CALL SEARCH(CLOSE,Ø,FUNIT,Ø)
C
C OPEN VARREC FILE
C
      CALL SEARCH(OPNRED,'VARREC',FUNIT,Ø)
C
C POSITION TO THE RECORD USING THE FILE POINTER IN ABSPOS AND
C READ THE RECORD SIZE INTO BUFF(1)
C
      CALL PRWFIL(PREAD+PREABS,FUNIT,PBUFF,1,ABSPOS,Ø)
C
      SIZE=BUFF(1)
      IF (SIZE.LT.1.OR.SIZE.GT.1ØØØ) GO TO 1ØØ
C
C READ THE REST OF THE BLOCK INTO BUFF(2)..BUFF(N)
C
      CALL PRWFIL(PREAD,FUNIT,PBUFF2,SIZE-1,Ø,Ø)
C
C WRITE THE RECORD TO THE TERMINAL
C
      WRITE(1,1Ø2Ø) RECNUM,SIZE
1Ø2Ø  FORMAT('RECORD'I6,' IS'I6,' WORDS AS FOLLOWS:')
      WRITE(1,1Ø3Ø) (BUFF(I),I=1,SIZE)
1Ø3Ø  FORMAT(1ØI7)
C
C CLOSE FILE AND RETURN TO PRIMOS
C
9Ø    CALL SEARCH(CLOSE,Ø,FUNIT,Ø)
      CALL EXIT
C
C RECORD SIZE ERROR
C
1ØØ   WRITE(1,1Ø4Ø)
```

```
1Ø4Ø  FORMAT('BAD RECORD SIZE')
      GO TO 9Ø
C
      END
```

```
C GPTRFL, CARLSON, JULY 16, 1974
C
C GPTRFL- GENERATE-POINTER-FILE
C PROGRAM TO GENERATE A FILE PTRFIL OF 2-WORD FILE POINTERS
C TO EACH LOGICAL RECORD IN FILE VARREC. VARREC CONSISTS
C OF LOGICAL RECORDS EACH OF WHICH CONSISTS OF A HEADER WORD
C THAT CONTAINS THE SIZE OF THE RECORD FOLLOWED BY THE DATA
C IN THE RECORD.
C
C RESTRICTIONS: RECORD SIZE MUST BE BETWEEN 1 AND 1000.
C               PTRFIL SHOULD NOT EXIST BEFORE RUNNING THE PROGRAM.
C               VARREC MUST EXIST IN THE CURRENT UFD.
C
      INTEGER FUNIT,PTRUNT,UERVEC(4),PUERVC,PUERV3,PSIZE,SIZE
C
C VARIABLE DEFINITONS
C FUNIT- CONTAINS 1, REFERS TO FILE UNIT 1 ON WHICH VARREC IS OPEN
C PTRUNT- CONTAINS 2, REFERS TO FILE UNIT 2 ON WHICH PTRFIL IS OPEN
C UERVEC- USER ERROR VECTOR, HOLDS ERRVEC OBTAINED FROM DOS
C SIZE- HOLDS SIZE OF LOGICAL RECORD
C PSIZE- POINTER TO SIZE
C PUERVC- POINTER TO UERVEC
C PUERV3- POINTER TO UERVEC(3)
C
C ROUTINES CALLED
C LOC,SEARCH,PRWFIL, GETERR,PRERR,EXIT
C
$INSERT KEYCOM
C
      DATA FUNIT,PTRUNT/1,2/
C
C
C INITIALIZE
C
      PUERVC=LOC(UERVEC(1))
      PUERV3=LOC(UERVEC(3))
      PSIZE=LOC(SIZE)
C
C OPEN VARREC FOR READING ON FUNIT
C
      CALL SEARCH(OPNRED,'VARREC',FUNIT,0)
C
C OPEN A NEW DAM FILE PTRFIL FOR WRITING ON PTRUNT
C
      CALL SEARCH(OPNWRT+NDFILE,'PTRFIL',PTRUNT,0)
C
C SET SIZE FOR FIRST TIME THROUGH LOOP. SIZE IS SET SO
C NO POSITIONING TAKES PLACE ON 1ST CALL TO PRWFIL. ERRVEC(3)
C AND ERRVEC(4) ARE SET TO FILE POINTER OF 1ST RECORD.
C
      SIZE=1
C
C POSITION TO NEXT LOGICAL RECORD OF VARREC. WE HAVE
```

```
C ALREADY READ ONE WORD OF RECORD, SO TO GET TO BEGINNING OF
C NEXT RECORD WE MUST POSITON FORWORD SIZE-1 WORDS.
C
10     CALL PRWFIL(PREAD+PREREL,FUNIT,0,0,SIZE-1,$90)
C
C GET FILE POINTER FROM ERRVEC
C
       CALL GETERR(UERVEC,4)
C
C FILE POINTER IS IN UERVEC(3) AND UERVEC(4). WRITE 2 WORD
C FILE POINTER INTO PTRFIL.
C
       CALL PRWFIL(PWRITE,PTRUNT,PUERV3,2,0,0)
C
C READ 1ST WORD OF NEXT LOGICAL RECORD INTO SIZE. 1ST WORD
C IS SIZE OF NEXT LOGICAL RECORD.
C
       CALL PRWFIL(PREAD,FUNIT,PSIZE,1,0,$100)
C
C IF SIZE OK, LOOP TO READ NEXT RECORD.
C
       IF (SIZE.GE.1.OR.SIZE.LE.1000) GO TO 10
C
C ERROR
C
       WRITE(1,1000)
1000   FORMAT('A RECORD HAS A BAD HEADER WORD')
       GO TO 110
C
C FILE ENDS IN MIDDLE OF A RECORD
C
90     CALL GETERR(UERVEC,1)
       IF (UERVEC(1).NE.'PE') GO TO 120
       WRITE(1,1010)
1010   FORMAT('FILE ENDS IN A PARTIAL RECORD')
       GO TO 110
C
C PRWFIL ERROR RETURN, CHECK TYPE.
C
100    CALL GETERR(UERVEC,1)
       IF (UERVEC(1).NE.'PE') GO TO 120
C
C
C FILE ENDS NORMALLY, CLOSE FILE AND RETURN TO PRIMOS
C
110    CALL SEARCH(CLOSE,0,FUNIT,0)
       CALL SEARCH(CLOSE,0,PTRUNT,0)
       CALL EXIT
C
120    CALL PRERR
       CALL EXIT
C
       END
```

APPENDIX D

OPERATING SYSTEM AND FILE SYSTEM MEMORY USAGE (SECTORED MODE PRIMOS II)

PRIMOS II occupies approximately nine sectors at the top of the available memory, plus a variable number of 448-word file unit buffers. Figure D-1 shows a typical memory map for a system with 16K of high-speed memory. PRIMOS III and IV take no part of the user's virtual address space.

Floating PRIMOS II

Three versions of PRIMOS II are supplied in the UFD DOS. These versions load PRIMOS II starting at locations '27ØØØ, '47ØØØ, and '67ØØØ. The bootstrap program selects the version of PRIMOS II that is nearest to the top of high-speed memory. The values in Figure D-1 may be increased accordingly to 2Ø,ØØØ and 4Ø,ØØØ locations to give an approximation of memory allocation for 32K and 64K systems. If desired, a particular PRIMOS II may be selected by manually setting the sense switches (refer to "BOOT" in the Computer Room User Guide (MAN 26Ø3)).

Other Locations

Sector Ø is reserved. On the Prime 1ØØ/2ØØ/3ØØ computer systems, locations Ø throug '177 are dedicated to the Prime CPU's register file and the vector locations for interrupt and DMX. Locations '2ØØ and above are used to store the cros -sector indirect address links generated by the loader, but the user may, with caution, use locations in this area.

For 16K configurations, locations '1ØØØ through '17777 may be used without restrictions, unless a symbol table is present. The high end limit is usually determined by the start of the loader, which may be memory-resident during loading. However, FORTRAN common may set the upper limit if it extends below 'Ø2ØØØØ.

FORTRAN COMMON overlaps part of the area that can be occupied by PRIMOS II file unit buffers. Up to three SAM file units can be open at a time without the risk of writing over part of COMMON.

Default location of FORTRAN COMMON is the top of the loader extending down in memory. There are two implications:

1. COMMON cannot be loaded with "BLOCK DATA" statements.

2 Only three disk units may be open at any one time. (PRIMOS II restriction only).

This problem can be avoided through use of the loader's COMMON command,

which permits the moving of COMMON to a user specified location.

If a program is to be debugged with the aid of Trace and Patch (TAP), only two files can be open at a time. However, TAP can relocate itself elsewhere in memory if this is a problem. For information on TAP, refer to the Program Development Software User Guide.

Memory areas occupied by the PRIMOS II file unit buffers are listed in the following table:

Table D-1

Memory Areas and PRIMOS II File Units

Number Of Open File Units	Top of Available Memory		
	16K System	24K System	32K System
0	'26777	'46777	'66777
1	'26077	'46077	'66077
2	'25177	'45177	'65177
3	'24277	'44277	'64277
4	'23377	'43377	'63377
5	'22477	'42477	'62477
6	'21577	'41577	'61577
7	'20677	'40677	'60677
8	'17777	'37777	'57777
9	'17077	'37077	'57077
10	'16177	'36177	'56177
11	'15277	'35277	'55277
12	'14377	'34377	'54377
13	'13477	'33477	'53477
14	'12577	'32577	'52577
15	'11677	'31677	'51677
16	'10777	'30777	'50777

Notes:

1. 448 words for each SAM file open.

2. 896 words for each DAM file open.

3. There is a difference of octal 700 as the number of open file units increases. Users can estimate the correct figures if they know how much memory is

available and the number of open file units.

4. The above figures assume only SAM files. Up-to-date information may be gathered by the use of the STATUS and GINFO commands.

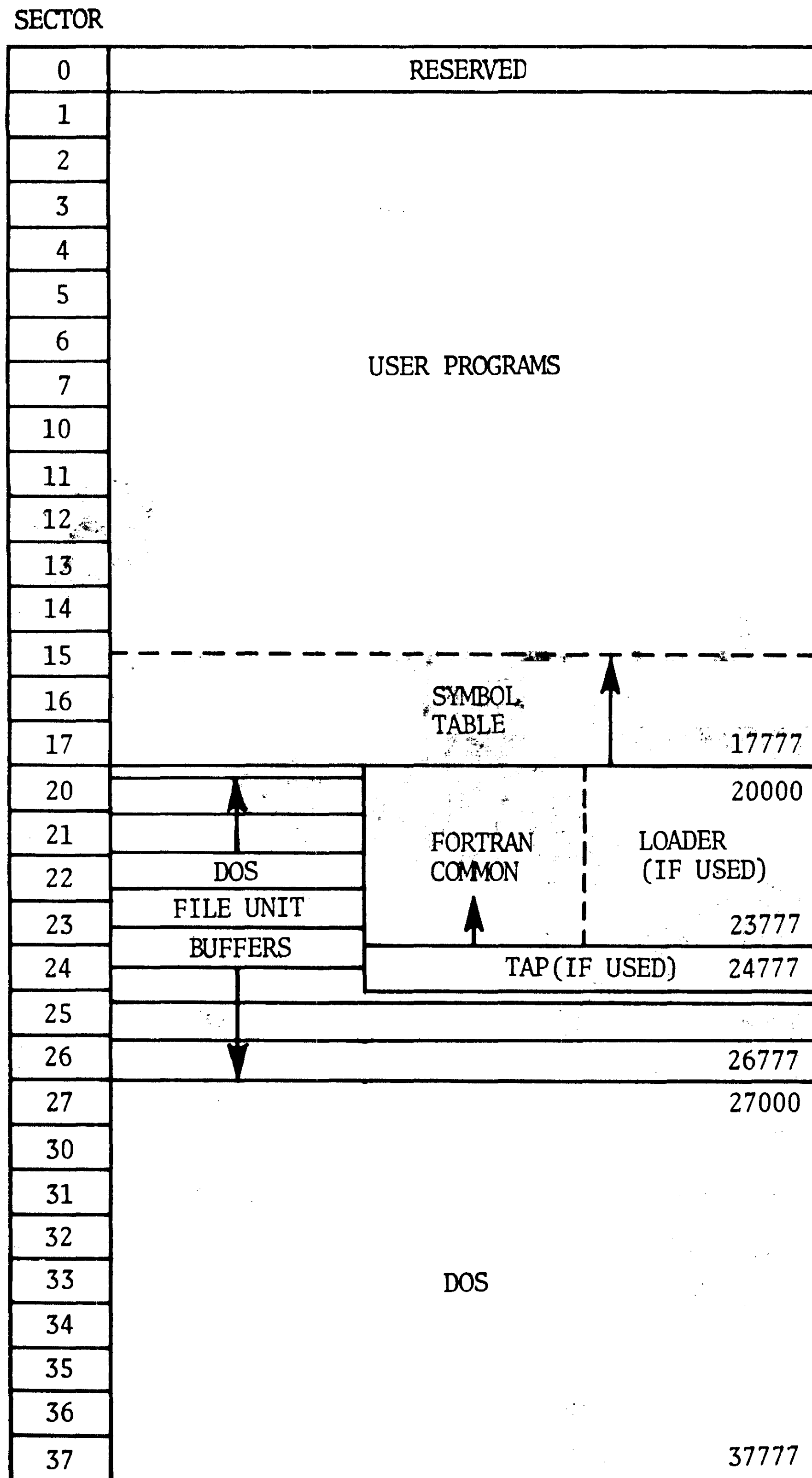


Figure 2-2. Memory Allocation in 16K System.

INDEX

ACCESS TIME, DISK B-5
APPLICATION EXAMPLES
ATTACH 3-4
ATTACH 4-7
BREAK$ 3-7
CALLING LOADING LIBRARY SUBROUTINES 3-1
CALLING SEQUENCE NOTATION 3-1
CMREAD 3-8
CNAME$ 3-9
COMANL 3-11
COMANL 3-13
COMEQV 3-13
COMINP 3-10
COMINP 3-12
COMMANDS, FUTIL 4-3
CONCEPTS, FILE SYSTEM 1-1
COPY 4-8
COPYDAM 4-9
COPYSAM 4-9
D$INIT 3-14
DELETE 4-12
DESCRIPTION OF FUTIL COMMANDS 4-3
DIRECTORIES 2-5
DISK ACCESS TIME B-5
DSKRAT FORMAT A-4
ERRSET 3-15
EXIT 3-16
FILE ACCESS 2-10
FILE AND HEADER FORMATS A-1
FILE FORMAT A-1
FILE HANDLING SUBROUTINES 2-8
FILE SECURITY B-5
FILE STRUCTURE (TREENAMES) 4-1
FILE STRUCTURES 2-1
FILE SYSTEM 2-1
FILE SYSTEM CONCEPTS 1-1
FILE SYSTEM PERFORMANCE B-4
FILE SYSTEM SUBROUTINES
FILE SYSTEM TERMINAL I/0 SUBROUTINES 3-2
FILE SYSTEM, USING 1-2
FILE UTILITY COMMAND (FUTIL) 4-1
FILES AND DISK STRUCTURES 2-6
FILES, TYPES OF 2-1
FORCEW 3-16
FORMAT, FILE A-1
FORMAT, UFD A-2
FORMATS, FILE & HEADER A-1
FROM 4-5
FROM* 4-6
FUTIL 4-1
FUTIL RESTRICTIONS 4-17
GENERAL INFORMATION B-1
GETERR 3-16
GETWRD 3-17
GINFO 3-17
INTRODUCTION 3-1
LISTF 4-14
NAMEQV 3-18
NUMBER INPUT OUTPUT 3-55
OPERATING SYSTEM USER INTERACTION 2-12
PRERR 3-18
PRWFIL 3-19
PUTC 3-24
QUIT 4-4
RDCOM 3-24
RECYCL 3-24
RESTOR 3-25
RESTRICTIONS, FUTIL 4-17
RESUME 3-25
RREC 3-26
SAVE 3-29
SEARCH 3-30
SECURITY, FILE B-5
STRUCTURES, FILES & DISKS 2-6
SUBROUTINES, FILE HANDLING 2-8
SUBROUTINES, LIBRARY 3-1
SUBROUTINES, TERMINAL I-O 3-2
T$CMPC 3-39
T$LMPC 3-41
T$MT 3-43
T$SLC 3-44
T$VG 3-48
T1IN 3-36
TIMDAT 3-38
TIOU 3-36
TNOU 3-36
TNOUA 3-37
TO 4-7
TOOCT 3-37
TRECPY 4-10
TREDEL 4-13
TREENAMES 4-1
TYPES OF FILES 2-1
UFD FORMAT A-2
UFDCPY 4-11
UFDDEL 4-13
UPDATE 3-57
USE OF PRIMOS FILE SYSTEM C-1
WREC 3-58