PTDOS

Processor Technology Disk Operating System

User's Manual

Describes PTDOS 1.5

Processor Technology Corporation

7100 Johnson Industrial Drive
Pleasanton, CA 94566
Telephone (415) 829-2600

Second Edition, First Printing, October, 1978
Manual Part No. 731029

IMPORTANT NOTICE

TABLE OF CONTENTS

PART I

PREFACE

This manual describes PTDOS 1.5, the result of continued evolutionary development of the previous release, PTDOS 1.4. In addition to the correction of all known bugs, a number of new features have been added to PTDOS:

*The HELP, DCHECK, and XREF commands are new. + MEDIT
*Extended Disk BASIC replaces BASIC/5.
*The DISKCOPY, FREE?, and $PR commands have additional capabilities.
*The system may be configured with a permanently open log file, to which all commands are echoed. This addition is reflected in the CONFIGR, SET, and SYST commands.
*The console device driver is initialized differently.
*Several new device driver Control/Status operations have been defined.
*Changes in wording have been made in many of the error messages output by the Explain Error Utility.

The video-oriented editor (EDIT) has several new features:

*A new command (TS) has been added to set and clear tab stops.
*The TAB key (CTRL/ I) moves the cursor to the next tab stop.
*The LOAD key (CTRL/ L) causes the previous string search to be continued.
*Two pattern deletion commands (PA D and PA DS) have been added.
*A new command (IF) has been added to allow files to be inserted into a file that is being edited.

The assembler (ASSM) has several new features:

*TITL, PAGE, and ASCZ pseudo-operations have been added.
*Labels may have lower case letters in any position.
*Operation mnemonics and register names may be lower case.
*An octal number may be followed by an O or a Q.
*Binary numbers are assumed to be right-justified, like octal, decimal, or hexadecimal numbers.

One change has been made in the debugger (DEBUG):

*The default output driver for DEBUG is now the PTDOS console output driver, rather than the internal VDM driver.

SECTION 1

OVERVIEW

1.1 INTRODUCTION

PTDOS, the Processor Technology Disk Operating System, is an operating system for 8080-based computers equipped with the Helios II Disk Memory System.

This is a software reference manual. A reader interested in a more detailed discussion of hardware than is available under the heading HARDWARE ENVIRONMENT, below, is referred to the Helios II Disk Memory System User's Manual. A reader interested in a tutorial introduction to the system is referred to Appendix 1 of the present manual.

1.2 SYSTEM OVERVIEW

An operating system is a program that provides an interface between the user and the computer hardware. This interface has two major components. First, it allows the user to specify conveniently the programs that will be executed and to supply parameters that control the functioning of those programs. Second, it provides a file system to facilitate management of the user's data.

In PTDOS these facilities have a very general implementation. System commands are simply program names presented to a Command Interpreter, which will load and run any file whose name appears in its input, provided that the file contains executable code. PTDOS files usually reside on a diskette, but any device connected to the system may be treated in the same way as a disk file.

Almost all system services may be accessed in two ways: by means of system calls from an assembly language program, or by means of the system programs supplied on the PTDOS diskette.

The following is a list of some of the features of PTDOS:

USER INTERFACE

*Any program name may be used as a command.
*Command Interpreter input and output may be redirected.
*Command files allow execution of a sequence of commands.
*Command macros allow parameterized command files.
*Commands may be echoed to a selected device or file.
*Commands may be logged on a permanently open system log file.

SYSTEM MANAGEMENT

ᵏSystem configuration may be changed by the user at any time.
ᵏSystem configuration is protected by a password.
ᵏThe system may be write-locked to protect data from being changed inadvertently.
*The user may create a startup file containing commands that will be executed automatically when the system is bootstrapped.

FILE SYSTEM

*Disk space is allocated and deallocated automatically as files are extended or shortened.
*Disk files may be accessed sequentially or randomly.
*A device-independent I/O system enables devices to be accessed as files.

MEMORY MANAGEMENT

*File buffers may be static or dynamic.
*Static buffers may reside in either system or user memory.
*User memory may be protected from Ø to a specified address.

ASSEMBLY LANGUAGE SYSTEM INTERFACE

*All primitive file operations are available as system calls.
*Three different methods of handling errors may be assigned to each of three error severity levels.
*A system Utility Handler allows programs to be constructed with multiple overlays.
*Standard system utilities may be accessed by a user program to display error messages or to obtain information about the files on a diskette.
*The console device may be accessed directly, one character at a time.
*Provision is made for interrupt-driven devices.
*A Parameter Scanner simplifies the processing of arguments by user programs.
*Commonly used system parameters are defined in files that may be included in a user program.

The commands and programs in PTDOS may be categorized as follows:

SYSTEM CONTROL

BOOTLOAD	Reload PTDOS from diskette.
HELP	Display information about command(s).
SYST	Display system parameters.
CONFIGR	Change system parameters on a diskette.
SET	Change system parameters in memory.
SETIN	Make named file the CI input file.
SETOUT	Make named file the CI output file.
OUT	Set console output to display or port driver.
FREE?	Report amount of free space on diskette.
OPEN?	Print name and number of each open file.
DO	Invoke command macro processor.
EXEC	Execute code at a specified address.
IMAGE	Write contents of memory to a file in image format.
ZIP	Fill memory with number.
$LST	Turn on PTDOS echo flag.
$NLST	Turn off PTDOS echo flag.
$WAIT	Wait for a carriage return or MODE SELECT.
$ESC	Check for a MODE SELECT.
$STOP	Return to the system from a macro file.
$REM	Identify a string as a remark.

PRIMITIVE FILE OPERATIONS

CREATE	Create a file on a diskette.
$CREATE	Create a file on a diskette.
KILL	Kill file(s).
OPEN	Open a file.
CLOSE	Close open file(s).
SPACE	Move the file cursor.
SEEK	Position the file cursor.
READ	Transfer contents of file to memory.
WRITE	Write contents of memory to a file.
ENDF	Endfile at current cursor position.
RANDOM	Create index block for file.
RENAME	Change name of file(s).
RETYPE	Change the type of a file.
REATR	Change protection attributes of file.

FILE MAINTENANCE PROGRAMS

FILES	Display list of files.
COPY	Copy contents of file(s) to another file.
DUMP	Display contents of file in hexadecimal and ASCII.
PRINT	Print file on the CI output file or the named file.
SAVE	Write one or more files to an archive file.
GET	Transfer file(s) from a file or diskette.
EXTRACT	Display load information; optionally, combine image segments.
BLDUTIL	Build or alter a utility file; list current utility numbers.

DISK MAINTENANCE PROGRAMS

DISKCOPY	Condition, format, copy, or verify a diskette.
DCHECK	Check structure of files on diskette.
RECOVER	Reclaim lost space on diskette.

PROGRAM DEVELOPMENT TOOLS AND LANGUAGES

MEDIT	Invoke screen-oriented machine language editor
EDIT	Invoke screen-oriented text editor.
EDT3	Invoke line-oriented text editor.
RNUM	Renumber lines of text file.
ASSM	Assemble an 8080 assembly language source file.
XREF	Generate cross-reference listing of assembly language file.
DEBUG	Invoke Debugger
BASIC	Invoke Extended Disk BASIC Interpreter.
FOCAL	Invoke FOCAL Interpreter.

GAME PROGRAM

TREK80	A video Star Trek game

1.3 HARDWARE REQUIREMENTS

The minimum hardware configuration for PTDOS is an 8080-based computer with at least 16K bytes of read-write memory, a terminal device, and a Helios II Disk Memory System. The code used by PTDOS to access files and provide system services is resident, occupying 12K bytes of memory from 9000 to BFFF Hex. Programs and data are loaded from diskettes. If the computer is not a Sol Terminal Computer, the Processor Technology VDM-1 Video Display Device is recommended for console output, although drivers are supplied for standard terminals.

1.4 ORGANIZATION OF THE MANUAL

The rest of the manual is organized as follows:

Section 2 describes the commands that will be almost the whole experience of PTDOS for users other than assembly language programmers. The syntax, purpose and special features of each command are explained.

Section 3 describes the Command Interpreter (CI), the interface that enables the user to communicate with the computer through the console. This discussion includes a list of the rules that govern the syntax of commands, as well as material about the selection of CI input and output files, and an explanation of some system parameters related to the operation of the CI.

Section 4 describes the command macro preprocessor (DO); this program substitutes user-supplied parameters for the variables in a macro command file, and causes the commands in the macro command file to be executed.

Section 5 contains detailed information about the PTDOS file system. In addition to a general description of the structure and properties of files, there are more technical discussions of disk allocation, file access and buffering, and data structures peculiar to some kinds of files, i.e., image files, utility files, text files.

Section 6 provides the information that a programmer must have in order to interface with PTDOS from an assembly language program. Included are discussions of memory management, system global parameters and entry points, error handling, and interrupt programming.

Section 7 describes the system calls that can be made from an assembly language program. The description of each call includes a discussion of the operation performed, its calling sequence, and possible error conditions.

Section 8 describes the system utilities that may be invoked by a program to display error messages or to obtain information about the files on a diskette.

Section 9 contains a detailed discussion of device drivers. This discussion includes a description of the system console driver and of the cassette tape driver that is supplied with PTDOS.

Section 10 is a list of error messages and their meanings.

Part 1 of the manual concludes with three appendices: a tutorial introduction to PTDOS, information about loading PTDOS, and a summary of the system commands.

Part 2 of the manual consists of full descriptions of the text editors, assembler, debugger, and FOCAL interpreter that are supplied with PTDOS.

An Extended Disk BASIC interpreter and the TREK80 video game are recorded on the same diskette as PTDOS, but are described in separate manuals.

SECTION 2

SYSTEM COMMANDS

2.1 INTRODUCTION AND CONVENTIONS

This section discusses each of the programs available as commands in PTDOS 1.5. The commands are arranged in alphabetical order for easy reference. The description of a command includes:

1) the name of the command, and a short description of its function,

2) a general statement of the syntax of the command, that is, what you should type to execute the command,

3) a brief description of the operation of the command,

4) a discussion of the "arguments" that follow the command name on the command line,

5) any additional details or notes that clarify the earlier brief description, and

6) examples, unless the command line consists simply of a command name.

All commands are interpreted by a program called the Command Interpreter (often abbreviated CI), which expects them to have a certain format and to follow other rules. The Command Interpreter is discussed at length in Section 3 of this manual. In order to use the PTDOS commands, there are a few rules that you must understand and follow:

1) Type the name of the command right after the PTDOS prompt (*). Follow the command name with a space, and then type the arguments, separating these by commas. Additional spaces may be inserted for clarity but are not required.

2) If you make a mistake while entering a command, you can use the DEL key to back up and erase the last character you typed. If you want to type the whole line again, press the CTRL and X keys simultaneously, and then enter the correct line. The CTRL and X keys are reflected on the Video Display as an exclamation point (!).

3) A unit specification (if it is not appended to a file name) or any argument that has the form " X=option(s)," where X is a code consisting of one or two letters and option(s) indicates the pertinent values, may appear anywhere in the list

of arguments; such an argument is "keyword delimited." Any other argument is "positional," and must appear in a certain position in the syntax of a command. If a positional argument is optional and you want to omit it from the command, type an extra comma to indicate the omission. (Actually, you need to do this only if you are including other positional arguments later in the command line.) For example,
ASSM PROG1,LIST,,+L means "assemble PROG1 with line numbers and send the listing to LIST but do not generate an object file." If you omit an argument, a default value for that argument will be used.

4) More than one command or program name may be typed on one command line as a list, as long every item except the last is followed immediately by a comma. Each command in the list is loaded, but only the last is executed. This way of entering more than one command applies only if none of the commands (except possibly the last) have arguments.

5) A command line must end with a carriage return. Multiple commands, with or without arguments, may be entered on the same command line if each command and its argument list is separated from the next command argument list by a semi-colon (;). None of the commands in a command line is executed until the carriage return is entered. (Note that if the commands are separated by semi-colons, the commands will be loaded and executed in sequence, whereas if they are separated by commas, only the last command in the series will be executed.)

6) A slash and a unit number can be appended to a filename to indicate which unit is to be searched for the file. Because a command is a file, any command name may have a unit number affixed to it. Thus, if you type FILES/1, the FILES command will be loaded from unit 1 and will generate a list of the files on the default unit. In the examples in this section, the system diskette is assumed to reside in the default unit. (If you have never changed the default unit number, it will be Ø, the leftmost unit in the drive.)

7) With the few exceptions noted, numerical arguments are assumed to be hexadecimal. To indicate that an argument has a different number base, follow the argument with a colon (:) and a letter from this list:

    H for a Hexadecimal number
    D for a Decimal number
    Q for an Octal number (O may be used, but Q is recommended)
    B for a Binary number

8) A file number must almost always be preceded by a # sign.

2.2 COMMAND DESCRIPTIONS

This section describes the individual commands. The following conventions will be used in all statements of command syntax:

1) Upper-case letters are literal, as are numbers and most marks of punctuation. In the command

*FREE? /1

the word FREE, the question mark, the space, the slash, and and the number 1 are actually to be entered on the keyboard or console. The asterisk (*) is the PTDOS prompt.

2) Lower-case letters indicate the category of item from which which the actual entry must be taken. For example,

filename

means that the name of a file should be entered on the console.

3) Optional arguments are enclosed in braces. For example,

{filename}

indicates that the name of a file is an optional argument.

4) The symbol * is the PTDOS prompt.

There are only two additional items that need explanation here:

If a command is marked SAFE, that command runs in the portion of memory occupied by PTDOS, and will not interfere with whatever programs and data you have stored in memory below PTDOS.

If a command is marked INTERRUPTABLE, that command will stop its operations if the MODE SELECT key (or CTRL - @) is typed on the keyboard.

THE COMMANDS

ASSM - Assemble an 8080 assembly language source file

*ASSM source,{list},{object},{error},{symbol},{S=options}

INTERRUPTABLE

OPERATION: The assembler translates a symbolic 8080 assembly language source program ("source code") into the binary instructions ("object code") required by the computer to execute the program. In addition to an object code file, the assembler can generate a listing file, an error file, and a file to which the symbol or cross-reference table is written.

ARGUMENTS:

source — is the name of the source code input file. This argument must be present; all others are optional.

list — is the name of the listing output file. If this argument is absent, no listing is generated. If the specified file does not exist, it is created with type '.' and block size 4CØH.

object — is the name of the object code output file. If this argument is absent, no object code is generated. If the specified file does not exist, it is created with type 'I' and block size 1ØØH.

error — is the name of the file to which lines with errors are written. (All lines, including those with errors, are written to "list," if that argument is present.) The default value for "error" is the console output file (file #1), unless "list" is #1, in which case there is no default error file.

symbol — is the name of the file to which symbol and cross-reference tables are written. If this argument is absent, no symbol or cross-reference table is generated. If an equals sign (=) is used instead of a filename, the table is written to the same file as the listing.

S=options — is a string of option specifiers. For those options that may be preceded by a + or -, the + is optional.

+A means that the source file is in ALS-8 format.
-A means that the source file is in text format.

If neither of these is specified, the assembler attempts to determine the file format by examining the first few lines of the file. If it fails, an error message is generated.

+L means that the source file has a line number in columns 1-4 of each line.
-L means that the source file has no line numbers.

If neither of these is specified, the assembler will examine the first few lines to determine whether the file has line numbers.

instructs the assembler to generate its own line numbers for use on the listing in place of those in the source file (if there are any).

P instructs the assembler to paginate output to the listing file.

X instructs the assembler to direct a cross-reference listing to the symbol file (if there is one). If a cross-reference table is written, a symbol table will not be written. The source code must either have line numbers or be assigned line numbers by the assembler. This cross-reference listing is generated very slowly; the XREF command performs the same function much faster.

Ø or 1 or 2 or 3 specifies the spacing on the listing:
Ø - no additional spacing
1 - 72 column output
2 - 8Ø column output (default)
3 - 132 column output

NOTES: The assembler is described in its own manual, entitled "ASSM: An Assembler for PTDOS 1.5" (See Part 2 of this volume.)

EXAMPLES:

```
*ASSM PROG1,#1,BIN
*ASSM PROG1,LIST,,ERRS,SYMBS,S=+A-#X-P
*ASSM ERRSONLY
```

BLDUTIL - Build or alter a utility file; list current utility numbers

```
*BLDUTIL utilfile{,I{number}=filename}{,D=number{,number,...}}
         {,S=L}
```

OPERATION: Modules are added to or deleted from the named utility file according to the arguments entered on the command line. BLDUTIL will create the utility file if it is nonexistent, and will build its directory if it is empty. More information about building and using utility files can be found in Section 6.4.5 of this manual. The format of utility files is described in Section 5.

ARGUMENTS:

Arguments are handled in the order that they appear. There are three possible arguments, any of which may be repeated on the command line: "utilfile" is the name or number of the utility file whose contents will be affected.

I{number}=filename

will cause the named file to be inserted into the utility file. The named file must be in image format and have a starting address. If "number" is present, the file will be as that module in the utility file. If there already is a module #number, BLDUTIL will ask whether you intend to replace the existing module; if your answer is Y, the existing module will be replaced by "filename." If "number" is not present, the file will be inserted at the first available position in the utility file directory.

D=number{,number...}

will cause module #number to be deleted from the utility file.

S=L

will cause a listing of the numbers of all existing modules to be printed on the console.

EXAMPLES:

*BLDUTIL NEWTIL,I=POTTS,I6=ZAP,D=3,S=L,IØ=ZOT,D=3,4,5

BOOTLOAD - Reload PTDOS from diskette

*BOOTLOAD

OPERATION: This command writes PTDOS into memory from diskette, overwriting the version that was in memory, without using the BOOTSTRAP loader program (see Appendix 2).

ARGUMENTS: This command has no arguments.

NOTES: The System Global Area contains a switch called the "verbose switch." A user is able to change the setting of this switch with the SET SW=H or CONFIGR command. If the switch is on when the BOOTLOAD command is given (or when the system is loaded from SOLOS/CUTER - see Appendix 2), the Command Interpreter will immediately execute the commands in the file START.UP on the system diskette. START.UP is a text file; you can edit it to include any commands that you want the CI to execute when the system is bootstrapped. For example, if the START.UP file is altered so that it contains the command to execute BASIC, the BASIC interpreter will be invoked automatically when PTDOS begins execution. If the verbose switch is not on when the system is loaded, the Command Interpreter will receive its first command from the console input device.

CLOSE - Close open file(s)

*CLOSE #fnum{,#fnum....,#fnum}
*CLOSE /{u}

OPERATION: In the first form of the command, the specified files are closed. If the second form does not include a unit number, all open files are closed, except the permanently open files; if a form does include a unit number, all open files on the given unit are closed except the permanently open files.

ARGUMENTS:

#fnum is the number of a file to be closed. The # is optional. If the file is not open, no error will be reported and no harm will be done.

u is the number of the disk drive unit that contains the the files to be closed. (Remember that units are numbered starting with Ø.)

/ closes all open files

NOTES: File numbers are assumed to be given in Hexadecimal; to specify a decimal number, follow the number with a colon (:) and a D, e.g., CLOSE #38:D will close file number 38 Decimal (26 Hexadecimal).

Initially, the permanently open files are #Ø (console input), #1 (console output), and #2 (the system utility file). If you have used the CONFIGR command to establish SYST.LOG (the system log file), that file will also be permanently open as file #3.

CONFIGR - Change system parameters on a diskette

*CONFIGR {/u,}password

INTERRUPTABLE

OPERATION:

The CONFIGR command allows the PTDOS user to change system parameters on the diskette, but not in memory. This command differs from SET, in that SET changes parameters in memory, but not on the diskette. The changes made by CONFIGR will therefore not take effect until the system is loaded again from the diskette.

ARGUMENTS:

/u — is the unit on which parameters are to be examined and perhaps changed. If this argument is absent, the default unit is used.

password — must match the password recorded on the named or default diskette; otherwise, the configuration of the diskette may not be changed. Initially the password is PTDOS, but this command may be used to change it. If a null password (consisting of all binary zeros) is created, the command may be used subsequently without a password.

NOTES: When CONFIGR begins execution, each parameter is except the password displayed along with its current value. A new value can then be entered for the parameter. If no new value is desired, type a carriage return to leave the parameter with its current value. Enter MODE SELECT (or CTRL-@) at any time to leave CONFIGR and return to the Command Interpreter. All input may be in upper or lower case.

The meaning of the parameters is explained in the discussion of the System Global Area in Subsection 6.3 of this manual. Some parameters are described in relation to the SET command.

CONFIGR will display:	Enter a carriage return or:
Disk name:	up to 8 characters, none of which may be control characters
Change password?	Y if you want to change it. Then type a new password. Entering a carriage return as the new password makes the password null (see explanation above).
Maximum number of units:	number from 1 to 8
User memory protect:	hexadecimal address (the lowest address into which programs or data may be loaded)
Console read char routine addr:	hexadecimal address
Console write char routine addr:	hexadecimal address
Console test char routine addr:	hexadecimal address
Lowest address of buffer area:	hexadecimal address less than 9001.

Interrupt flag:	On or off.
Echo enable:	On or off.
Disk write lock:	On or off.
Character upshift:	On or off.
Read-back check:	On or off.
Binary console I/O:	On or off.
System log:	On or off.
Verbose:	On or off.
Maximum files open:	decimal number from 7 to 255

EXAMPLES:

```
CONFIGR /1,PTDOS
```

COPY - Copy contents of file(s) to another file

```
*COPY infile,outfile{,S={A}{-E}}
*COPY O=file{,S=-E},infile1{,infile2...}
```

INTERRUPTABLE

OPERATION: In the first form of the command, data is copied from the input file to the output file until an end-of-file is encountered on the input file. In the second form, zero or more input files are concatenated and written to the output file. An input file is never modified by COPY.

ARGUMENTS:

infile,infile1,or infile2 is the name or number of a file from which data is copied. In the second form of the command, the named files are copied in the order that they appear in the command line.

outfile or O=file is the name or number of the file to which data is copied. If the file does not exist, it is created.

S=A means that the infile is appended to the outfile in the first form of the command. Otherwise the outfile is overwritten by the infile.

S=-E means that an endfile operation will not be performed on the output file after all data has been copied. Otherwise an endfile operation will be performed. (See the ENDF command, below.)

NOTES: The input file and the output file must not be the same, although a file may be copied to a file which has the same name but resides on a different unit.

This command may not be used if it would shorten a file open under more than one file number; such a file may only be lengthened if designated by the first number under which it was opened.

EXAMPLES:

```
*COPY MAN,MAN/1
*COPY TAIL,HEAD,S=A-E
*COPY O=MAN,MAN0,MAN1,MAN2,MAN3
```

CREATE - Create a file on a diskette

*CREATE filename{,{type}{,blocksize}}

SAFE

OPERATION: A file with the specified name is created on the specified unit with the specified type and block size. If a file with this name already exists, an error message is printed.

ARGUMENTS:

filename — is the name of the file to be created.

type — is the type to be assigned to the file. The default value for "type" is '.'

blocksize — is the hexadecimal block size to be assigned to the file. The default block size is 4C0H.

EXAMPLES:

```
*CREATE BIGFILE,M,9C0
*CREATE MYFILE/1
```

DBASIC - Invoke Extended Disk BASIC Interpreter

*DBASIC

OPERATION: This is a special adaptation of BASIC (Beginner's All-Purpose Symbolic Instruction Code) for use with PTDOS and the Helios II Disk Memory System.

ARGUMENTS: DBASIC has no arguments. An argument may be included in the command that executes the initialized version of BASIC (see NOTES, below); that argument is the name of a file containing a BASIC program. The named file is loaded and executed immediately, whether it was saved in text or semi-compiled form; if the file does not contain a BASIC program, an error will be reported.

NOTES: This program requires initialization; when DBASIC is first executed, the user is asked to supply certain information for use in this initialization. For example, should certain functions be deleted? The initialized version of the program should then be saved on the diskette under a name other than DBASIC; that name will then be the command to execute BASIC. The features and use of BASIC are described in the Extended Disk BASIC User's Manual, in a separate volume.

EXAMPLES:

*DBASIC
*BASIC PROGRAM (assuming BASIC is an initialized version of DBASIC)

DCHECK - Checks structure of files on diskette

DCHECK {/u}

INTERRUPTABLE

OPERATION: The directory is loaded from the specified or default diskette. If no error occurs, DCHECK checks each file by opening it, spacing through it, and closing it before proceeding to the next file. If an error occurs while a file is being checked, the file name and an error message are listed on the CI output file, and DCHECK goes on to check the next file.

The space operation requires that a buffer be allocated; therefore in order for all files to be checked, GLLOW must be set low enough to provide sufficient buffer space for the file that has the largest block size. If DCHECK attempts to open a file that has a block size larger than the available buffer space, an error message will be generated and DCHECK will go on to check the next file.

ARGUMENTS:

/u specifies the unit on which files are to be checked. If this argument is not given, the default unit is used.

EXAMPLE:

*DCHECK /1

DEBUG - Invoke 8080 Debugger

*DEBUG {arguments}
*DEBUG3 {arguments}

OPERATION: This command is an aid for debugging machine language programs. The debugger operates by permitting as many as fifteen breakpoints to be set in the program being examined. When the program is executed under control of DEBUG, it will stop at each breakpoint address so that CPU registers, flags, and specified memory locations may be examined and modified.

There are two DEBUG programs available for use; these versions are identical, except that they run at different memory locations.

DEBUG3 is loaded and executed at 3000H.
DEBUG is loaded and executed at 5000H.

Either program requires just over 4K bytes of memory.

ARGUMENTS: No arguments are read by the debugger. If arguments are entered on the command line, they are accessible to the next program that tries to read arguments from that line. For example, if the command line is

*FILES,DEBUG S=-I

and the FILES program is executed from within DEBUG, the S=-I parameter will be read by the FILES command.

NOTES: The debugger is described in its own manual, entitled "DEBUG: A Debugger for PTDOS 1.5." See Part 2 of this volume.

EXAMPLES:

*TESTPROG/1,DEBUG #4,ABC,300

(Load TESTPROG from unit 1 and enter the debugger. TESTPROG must read the arguments from the command line.)

DISKCOPY - Condition, format, copy, or verify a diskette

*DISKCOPY {/}from,{/}to{,S=-W}
*DISKCOPY /u,S={C}{F}{V}{-W} (only one of C, F, and V)
*DISKCOPY {/}unit1,{/}unit2{,S={{V}{-W}}

INTERRUPTABLE:

OPERATION: Each form of this command will perform one of the following functions:

- Condition a new diskette for use by the Helios II Disk Memory System
- Format a diskette for use by PTDOS (or erase the diskette
- Copy the contents of one diskette onto another
- Verify that the contents of one diskette are the same as the contents of another
- Verify that all data on a diskette can be read

ARGUMENTS AND NOTES:

CONDITION A NEW DISKETTE - DISKCOPY /u,S=C

This command writes the proper control signals onto the diskette in the specified unit. Until a diskette has been conditioned, the Helios hardware cannot be used to read from it or write to it. CONDITION A BRAND NEW DISKETTE BEFORE PERFORMING ANY OTHER OPERATION ON IT.

FORMAT A DISKETTE - DISKCOPY /u,S=F

This command writes necessary files onto the diskette in the specified unit, so that PTDOS may use the diskette as a "data disk." If the diskette being formatted already has information on it, that information will be erased; thus, this command may be used to erase a diskette. A diskette need not be conditioned prior to formatting.

COPY ONE DISKETTE TO ANOTHER - DISKCOPY /from,/to,{S=-W}

This command makes a track for track copy of the diskette in the "from" unit on the diskette in the "to" unit. If the destination diskette is brand new, it must be conditioned before this command is given. It is not necessary to format a new diskette before copying to it.

COMPARE TWO DISKETTES - DISKCOPY /unitl,/unit2,S=V

This command compares the data on the diskette in unitl with the data on the diskette in unit2. If a difference is found, a message is printed and the operation is discontinued.

TEST READABILITY OF A DISKETTE - DISKCOPY /u,S=V

This command reads every byte on the diskette twice and compares the two bytes. If a byte cannot be read, or if the two readings of the byte do not match, a message is printed and the operation is discontinued.

THE -W OPTION may be added to the S arguments of any of the above commands. If this option is not present, the command will require a carriage return between the time that it is first entered and the time that its function is actually performed; this feature is useful, because it enables the user to make a copy of a diskette not in the drive when the command is given. At any time before the carriage return, diskettes may be removed from the drive and replaced with other diskettes. In general, the -W option is used in situations in which user intervention is not desired: for example, in a command file to be executed with the SETIN command or the DO macropreprocessor (see Section 4). This option also affects the heading of READ errors (see below).

ERRORS AND LIMITATIONS:

It is not possible to format or copy onto disk unit Ø or the default unit. A diskcopy from a unit to itself is also unacceptable.

WRITE ERRORS will result in a retry of the write operation. The retries will continue until a successful write occurs or the operation is aborted with the MODE key. A write error is an indication of a hardware problem; the diskette, the disk drive, or the controller electronics may be at fault.

READ ERRORS are handled according to the setting of the W option. If the -W is set, the error will be displayed on the console and the bad sector(s) will be ignored. Otherwise the program will ask if a retry is desired and will continue to try until an "N" answer is given to that enquiry.

EXAMPLES:

```
*DISKCOPY /Ø,/1
*DISKCOPY /1,S=C
*DISKCOPY Ø,1,S=V-W
```

DO - Invoke command macropreprocessor

*DO {O=outfilename,}{S=options,}infilename{,parameters}

OPERATION: This program enables the user to create versatile macro command files. It allows variables to be inserted in a file to represent parameters that will be supplied in the command to execute the macro; it also provides for conditional substitutions based on the presence or absence of expected parameters in the DO command line.

ARGUMENTS:

infilename	is the name of the macro input file. This should be a standard EDIT or EDT3 text file, or a text file in ALS-8 format.
O=outfilename	specifies the name of the first output file to be generated by DO. If no such file exists, it will be created with type "$" and block size 100H. The default output file is $DO.CMD0.
S=options	Options are represented as a string following the (=) sign.
X or +X	dictates that the output file be executed immediately afters its construction. This is the default condition if the O argument is omitted from the command line.
-X	dictates that the output file not be executed immediately after its construction. This is the default conditon if the O argument is included in the command line.
A or +A	means that the input file is in ALS-8 format.
-A	means that the input file is in standard text format.
parameters	are the actual parameters to be substituted for the variables in the input file.

NOTES: Section 4 of this manual describes the DO macropreprocessor.

EXAMPLES:

```
*DO O=ZOO,ANIMALS,ZEBRA,ELEPHANT,17
*DO MAKE$,1,2,T
```

DUMP - Display contents of file in hexadecimal and ASCII

*DUMP file{,addr1{,addr2 or >count}}

INTERRUPTABLE

OPERATION: The contents of all or part of the specified file are displayed on the console output file in hexadecimal and ASCII. A non-printable character is represented by a period (.) in the ASCII part of the dump.

ARGUMENTS:

file	is the name or number of the file whose contents are to be dumped.
addr1	is the starting address of the dump.
addr2	is the ending address of the dump.
>count	indicates the number of bytes to dump, beginning with addr1. "count" should not exceed the number of bytes between addr1 and the end of the file.

NOTES:

If no address or count is given, the file will be dumped from beginning to end. If the second address or count is not given, the file will be dumped starting at addr1 and continuing to the end of the file. If addr2 and >count are both specified, PTDOS will display the message "DUMP ERROR: IMPROPER ARGUMENTS," and control will return to the Command Interpreter.

EXAMPLES:

```
*DUMP GOOP
*DUMP POTTS,5,100
*DUMP KNOCK,,>50:D
```

EDIT - Invoke screen-oriented text editor

*EDIT infile{<A>}{,{outfile{<A>}}{,top of memory}}

OPERATION: This command permits the input file to be edited. If no output filename is given, the edited file is written over the input file at the conclusion of editing. If an output filename is supplied, the input file is left unchanged and the edited file is written to the designated output file.

ARGUMENTS:

infile — is the name or number of the file to be edited. If the file does not exist, EDIT will ask whether it should be created. If the answer is Y, the file is created with type "." and block size 4CØH. A text file is expected to consist of text lines; a line consists of no more than 64 characters and ends with a carriage return (which does not count as one of the 64 or fewer characters).

outfile — is the name or number of the file to which edited material will be written. If that file does not exist, it is created with a type of "." and a block size of 4CØH.

<A> — specifies that the file is to be read or written as a file in ALS-8 format. The brackets are literal.

top or memory — determines the end of the text area allowed by EDIT. If this argument is not included in the command, either the highest good memory address or the lowest PTDOS system address (whichever of these is lower) will be used.

NOTES: This editor is for use with Processor Technology Sol Systems or other systems equipped with the VDM-1 display module. EDIT is described at length in its own manual, entitled "EDIT: A Text Editor for PTDOS 1.5."

EXAMPLES:

```
*EDIT MAILIST/1<A>,MAILIST/1 (notice conversion from ALS-8)
*EDIT KNOT,KNIT
```

EDT3 - Invoke line-oriented text editor

```
*EDT3
```

OPERATION: This command allows one or more files to be created, edited, and written to other files. The program allows editing on character, line, string, and page levels.

ARGUMENTS: The EDT3 command has no arguments. Control is passed to the program, which contains commands to regulate input and output.

NOTES: This editor loads in low memory and requires at least 8K of memory for proper operation. The EDT3 program is described at length in its own manual, entitled "EDT3: A Text Editor for PTDOS 1.5." (See Part 2 of this volume.)

ENDF - Endfile at current cursor position

ENDF fnum{,}

SAFE

OPERATION: The designated file is endfiled at the current cursor position. All data following this point is destroyed, and any disk space formerly occupied by that data is released for further use. If the file has an index for random access, the index is updated.

ARGUMENTS:

fnum is the number of the file that will be endfiled.

* means that the file should be left open at the conclusion of the operation. The number of the open file is displayed on the CI output file.

NOTES: This operation cannot be performed if it would shorten a file that is open under more than one file number.

EXAMPLES:

```
*ENDF #3
*ENDF #5,*
```

EXEC - Execute code at a specified address

*EXEC address

SAFE

OPERATION: This command transfers control to the specified address, which presumably contains either the first instruction of a program or a reentry point to that program.

ARGUMENTS:

address is the address to which control is to be transferred.

NOTES:

The operation of this command is similar to a standard assembly language CALL instruction. A RET instruction may be used to return to the system, if the stack has been maintained and CXBUF has not been disturbed. (CXBUF is the command execution buffer; see Section 6.2.)

When control is transferred, the DE register pair points to the first character AFTER the mark of punctuation that follows "address."

EXAMPLES:

*EXEC 0010

EXTRACT - Display load information; optionally, combine image segments

*EXTRACT file{,S{-L}}

INTERRUPTABLE

OPERATION: The specified file, which must be in image format, is read. The length and load address of each image segment is printed on the console output file. The image file may also be made more compact.

ARGUMENTS:

file	is the name or number of an image file.
S	means that image segments that load contiguously and are contiguous in the file are combined, and the file is rewritten.
-L	causes the listing of load addresses to be suppressed.

NOTES: The binary output file written by the assembler is in image format with a maximum segment size of 100 Decimal bytes. The S option of EXTRACT may be used to make such a file more compact.

EXAMPLES:

*EXTRACT POTTS,S
*EXTRACT KNOCK
*EXTRACT FILE,S-L

FILES - Display list of files

*FILES {/u}{,T=type}{,S={-H}{-I}}{,strings}

INTERRUPTABLE

OPERATION: A list of files is printed on the CI output file. The arguments dictate which files will appear in the list; if a file matches more than one argument, it will still be listed only once.

Each entry in the list consists of a the name of a file, the file type, the number 256-byte sectors allotted to the file (one 4CØ block is equivalent to four 256-byte sectors), the block size, the file ID, the sector and track on which the first block of the file is recorded, the protection attributes of the file, and the location of the index block. (There is no index block if the file is not a random access file; also, the attribute field will be blank in many cases, because many files will not have any protection attributes.)

ARGUMENTS:

/u — specifies the unit whose directory is the source of the list. If this argument is not present, the default unit is used. Only files that exist on the specified or default diskette will be listed.

T=type — If this argument is present, only files of the given type will appear in the listing. Specify image type files by preceding the type with the letter "I"; specify types that are non-printing by preceding the Hexadecimal value of the type with a #. T=I#5 signifies a type whose hex value is 5; T=I5 signifies a type whose value is the ASCII character 5. System files have a type value of zero and should be specified with T=#Ø or T=I#Ø.

S={-H}{-I}
- -H means suppress column headings on the listing
- -I means list files even if they are information-protected.

strings — In this command and in several others, a string may be used to represent all files whose names contain that string, or contain the string in a specific position in the name, for example, at the beginning. If one or more of these arguments are present in the command, only files whose names are identified by the string(s) will appear in the list. Otherwise, the names of all files that meet the other option requirements will appear. Strings may be typed in upper or lower case, i.e., "NAME" and "name" designate the same file.

String arguments may take any of the following forms:

string — may be any legal PTDOS file name, not including a unit number. If a file with this name exists, it will be included in the list.

string> causes all files whose names begin with the string to be included in the list.

<string causes all files whose names end with the string to be included in the list.

<string> causes all files whose names include the string to be included in the list.

EXAMPLES:

*FILES /1,S=-I,T=#ØØ

List all files of type ØØ, whether or not they are information-protected.

*FILES <AID>,<AER,KNOCK

List the file called KNOCK, and also any files that contain the string "AID" or end with the string "AER."

*FILES /1

List all files on unit 1, except information-protected files.

*FILES S=-I-H,SYSGLOBL

List the file SYSGLOBL if it is present on the default unit. The information will be listed even if the file is information-protected, and no column headings will be printed.

FOCAL - Invoke FOCAL Interpreter

*FOCAL

OPERATION: FOCAL is a high-level language described in the 8Ø8Ø FOCAL User's Manual in Part 2 of this volume.

FREE? - Report amount of free space on diskette

*FREE? {/u}{,blksize}

SAFE

OPERATION: This command displays the decimal number of sectors free on the specified or default diskette. If a block size argument is supplied, the command will display the number of free blocks of the specified size, if optimum, or the next higher optimum size. Optimum block sizes are discussed in Section 5 of this manual. If no block size is supplied, the command will display the number of free 256-byte sectors.

ARGUMENTS:

/u — is the unit containing the diskette to be examined. If this argument is absent, the diskette in the default unit will be examined.

blksize — must be a hexadecimal number between 1 and FFF. Otherwise, the program prints an error message.

NOTES: This command operates by reading the FREE SPACE MAP that PTDOS maintains on each diskette. This map is discussed in Section 5 of this manual.

GET - Transfer file(s) from a file or diskette

*GET I=/u or file {,/u}{,T=type}{,S=options}{,strings}

OPERATION: GET is used to retrieve files saved with the SAVE command; it can also be used to transfer one or more files from one data diskette to another data diskette. The files are not actually deleted from the archive file.

ARGUMENTS:

I=file or /u — If the equals sign (=) is followed by "file," that option specifies the name or number of the archive file from which the files are to be taken. If the equals sign is followed by a unit number, all files that satisfy the name and type requirements are copied onto the unit given in the {,/u} argument, or onto the default unit. If the files being "gotten" do not already exist on the output unit, they are created with the type, block size, and attributes of the corresponding file on the input unit.

Only one I argument may be present.

T=type dictates that only files of the specified type will be retrieved.

strings have the same significance as in the FILES command, above.

S=options

N makes it impossible to retrieve a file if a file with the same name appears on the output unit. (If the file exists as part of a SAVE file on that unit, it may be retrieved, because its name no longer appears in the diskette directory.)

-L dictates that there be no listing of the names of files as they are retrieved.

R inhibits the retrieval process, so that the command produces only a list of the files that would have been retrieved if the R option had not been selected.

NOTES: This operation may not be performed if it would result in shortening a file open under more than one file number. A file may be extended only if it is designated by the number under which it was first opened.

EXAMPLES:

*GET /1,I=CTAPE1,<PAY>,<OLL

GET all SAVEd files that contain the letters "PAY" or end with the letters OLL. Files are transferred from the default unit to unit 1.

*GET I=FROM/1,T=G,S=N

GET from FROM/1 to the default unit all files of type "G". If any of these files already exist on the default unit, do not GET the corresponding file onto that unit.

*GET I=/1,/Ø,E>

GET from unit 1 to unit Ø all files whose names begin with the letter "E."

HELP - Display information about command(s)

*HELP {command name}{,command name....}

OPERATION: This command displays the syntax of the named command, gives a short description of its function, and lists available options with their meanings.

ARGUMENTS:

command name may be the name of any PTDOS command.

If no arguments are present, HELP will display more information about itself.

NOTES:

The HELP program utilizes a large data file called HELP:D on the system diskette.

EXAMPLES:

```
*HELD
*HELP GET
*HELP HELP
```

IMAGE - Write contents of memory to a file in image format

*IMAGE file,{!blksize,}segl,seg2{,:seg3}....,segn{,sa}

SAFE

OPERATION: This command writes data from memory to an image file, that is, to a file that can later be loaded and executed if its name is typed after the PTDOS prompt. The structure of an image file is discussed in Section 5.7.3.

ARGUMENTS:

file is the name or number of the image file to which data be written. If that file does not exist, it created with type "I" and the indicated or default block size. In the rest of this discussion, "block" refers to a portion of memory, rather than to a physical sector on the diskette.

!blksize is the block size to be assigned to the named file if that file must be created. If this argument is absent, the file will be created with block size 256. If the named file already exists, this argument will be ignored. The ! is literal.

Each "seg" is a pair of numbers with one of three forms:

1) nl,n2 where both n's are numbers and nl is smaller than or equal to n2. The block of memory between address nl and address n2, inclusive, will be written to the specified file as a single segment.

2) nl,>count where nl and "count" are numbers. The contents of "count" bytes, starting at address nl, will be written to the file as a single segment. The > is required.

3) either of the above followed by ":n3" where n3 is preceded by a colon and is not equal to nl. n3 is the address at which this segment should be loaded, rather than at nl. The colon is required.

sa is the starting address for the image file. The value of "sa" is not restricted to the area which will be occupied by "file" when it is loaded. If this argument is included in the command, it must be the last item in the argument list. If this argument has not been supplied, control will pass back to the Command Interpreter after the file is loaded.

NOTES: There is no check for the type of the file to which the data is being written, but there is no reason to IMAGE data to a file that is not an image file.

EXAMPLES:

*IMAGE DATA 100,2C0

Write the contents of memory from 100 to 2C0 to a file called DATA on the default unit. If the word DATA is entered subsequently as a command, each byte of the image block will be loaded at the same location from which it was written by IMAGE, and control will return to the CI.

*IMAGE PROGRAM,!4C0,100,>8FC,24D

Write the contents of the 8FC bytes starting at 100 to a file called PROGRAM. When PROGRAM is loaded, control will pass to address 24D. If PROGRAM does not exist, it will be created with type 'I' and block size 9C0.

*IMAGE EXAMPLE,0,50,:4000,100

Write the contents of memory from 0 to 50 to a file called EXAMPLE. Thereafter, when EXAMPLE is loaded, the data will be loaded starting at address 4000, and control will pass to address 100.

*IMAGE EXAMPLE,0,177:Q,300,FFF,1200,>1024:D,4000,6FFF,:3000

Write the contents of memory from 0 to 177 octal, from 300 to FFF hexadecimal, from 1200H forward 1024 decimal bytes, and from 4000 to 6FFF hexadecimal, to a file called EXAMPLE. When EXAMPLE is loaded, each segment will be loaded into memory at the address from which it was written, except the last segment, which will be loaded starting at 3000H. Control will pass back to the CI.

KILL - Kill file(s)

*KILL filename{,filename...}

SAFE
INTERRUPTABLE

OPERATION: Each named file is killed by having its entry removed from the diskette directory and its disk space reclaimed. Each block of data belonging to the file is rewritten with useless data.

ARGUMENTS:

filename is the name of a file to be killed. Files to be killed need not all exist on the same diskette.

NOTES: After each file is killed, a message of the form:

filename IS KILLED

is printed on the CI output file. If a file you are trying to kill is nonexistent or protected against the KILL operation, the word "KILLED" will be replaced by "NONEXISTENT" or "PROTECTED," whichever is appropriate, and the program will continue to the next file named on the command line. A file cannot be killed if it is open when the command is given.

EXAMPLES:

*KILL POTTS/1,POTTS,KNOCK

Kill the file POTTS on unit 1 and the files POTTS and KNOCK on the default unit.

OPEN - Open a file

*OPEN filename{,buffer address or T}

SAFE

OPERATION: The specified file on the specified unit is opened. The system assigns the file a number, which is displayed on the CI output file; this is the number by which you should refer to the file while it is open. A buffer is allocated for the file, and the first block of the file is read into the buffer. The file cursor is created and set to point to the first byte of the file. (Detailed information about file access and buffering can be found in Section 5 of this manual.)

ARGUMENTS:

filename	is the name of the file to be opened.
buffer address	is the address at which a static buffer should be allocated for the file. " buffer" address is the lowest address of the buffer; the highest is buffer address plus the block size of the file. A buffer can only be set in memory external to PTDOS, and only in unprotected memory. A buffer cannot be set at Ø or FFFFH.
T	indicates that dynamic buffering should be used. Otherwise static buffering is used.

If no second argument is given, the system will allocate a static buffer in the system-managed buffer area. It is not possible to specify both a buffer address and the letter T, because it is not possible for a user to assign a dynamic buffer outside the system-managed buffer area.

NOTES:

Several PTDOS commands allow you to designate a file either by its name or by its number. If you give the number of the file, rather than its name, the file must be open, because A FILE THAT IS NOT OPEN DOES NOT HAVE A FILE NUMBER. On the other hand, if you give the name of the file, the file is assumed NOT to be open; the command will open it, even if the file is already open, and assign it a second number and a second buffer in the system-managed buffer area. When a file is open under more than one file number, it is said to be "multiply open". Many of the commands that will open a named file will also close it when the desired operation is complete.

There is nothing inherently wrong about opening a file more than once. In fact, if you want to operate on two parts of a file simultaneously, you may have to open the file twice, so that you can have two cursors pointing to different bytes (possibly in different blocks) of that file. On the other hand, there are a few reasons to avoid opening files more often than necessary:

1) Every time that you open a file, buffer space is allocated for another block of that file. If you are not careful, you may overflow the system buffer area.

2) There are a number of operations that it is not possible to perform on a file that is open, or that is multiply open. In general, you do not need to concern yourself with this fact, because the syntax of the command will make it clear that a file must be designated by name only, or by number only. If you get an error message of the form:

<file> ALREADY OPEN or
<file> IS MULTIPLY OPEN

close the file, and you will probably be able to proceed.

EXAMPLES:

```
*OPEN MYFILE,453F  with buffer in user memory
*OPEN XFILE        normal static system buffer
```

OPEN? - Print name and number of each open file

*OPEN?

SAFE

OPERATION: This command lists the name, file number, and unit number of each open file. The unit number is represented as a two-digit number, e.g., FRITZIE/ØØ is on unit Ø.

ARGUMENTS: This command has no arguments.

NOTES: Device files do not have their names and unit numbers listed. They appear on the listing as:

D-FILE/99

Permanently open files (like the keyboard, the display, the system utility file, and possibly a system log file) are not listed at all.

OUT - Set console output to display or serial port driver

*OUT V or P

SAFE

OPERATION: The resident code of PTDOS has two output drivers. One is a video display driver, and the other is for the serial or parallel port. This command selects the specified driver as the console output file.

ARGUMENTS:

V or any word starting with V causes output to be directed to the Video Display driver.

P or any word starting with P causes output to be directed to the port driver.

NOTES: A detailed description of the console driver can be found in Section 9 of this manual.

EXAMPLES:

```
*OUT P
*OUT VDM
*OUT PRINTER
```

PRINT - Print file on the CI output file or the named file

*PRINT {args,}file{,{args,}file...}

OPERATION: PRINT lists each file on the CI output file, unless an argument specifies a different output file. The format in which a file is printed is determined by arguments that precede the name or number of that file in the command line.

ARGUMENTS:

Arguments are separated by commas and read sequentially. The arguments that precede a filename are presumed to apply to that file and to all files whose names appear later on the command line (unless one of the same arguments appears later with a different value, thereby altering the conditions for remaining files). If you omit an argument list because the desired parameters are already in effect, DO NOT INSERT A COMMA OR ANY OTHER CHARACTER to mark the omission.

file is the name or number of a file to be printed.

Within the argument list that precedes a filename, the order of these arguments is arbitrary:

O=file determines that output will go to the file whose name or number appears after the equals sign. The previous output file will be closed, unless it is the CI output file. (The CI output file will be left open, but it will receive no output.) If this argument is absent from the command line, all output will go to the CI output file. If "file" does not exist, it will be created.

H="string" declares that the given string, minus its delimiters (i.e., the quotation marks), is to be printed as a centered heading at the beginning of the next listing. Any character other than a blank, a semicolon, a carriage return, or a comma may be used as a delimiter; the same character must delimit the beginning and the end of the string, and must NOT appear within the string. If the output is to be paginated, the heading will be printed at the top of every page. If no string is specified (i.e., if this argument is not present, or if the first

non-blank character after the equals sign is a comma), no heading will be printed. The heading may not be more than one line long; line length is set by the R argument.

S=flags sets or resets the option flags whose single-character codes are given as a string after the equals sign. Each code may be preceded by a plus (+) or a minus (-); if neither of these signs appears, the meaning of a plus sign is assumed. When the PRINT program begins to be executed, none of the flags is set.

If the code for a flag is preceded by a '+', the flag is set: the option denoted by the flag will be in effect during the listing of all subsequent files; if the code for a flag is preceded by a '-', the option will no longer be in effect when subsequent files are listed. A symbol that is not a plus, a minus, or the code for a flag is ignored.

The flags and their codes are:

A - Subsequent files are in ALS-8 format (each line begins with a byte count).
D - Listings are to be double-spaced. If the output file is not a device file, an extra carriage return/linefeed is inserted into the output.
P - Listings are to be paginated. Page numbers start with 1 and can never exceed 9999. Each page number will be centered on the penultimate line of the page.
- Files are to be listed with a line number before each line of the original file. (For example, if a text file is comprised of 64-character lines and a 30-character page width is specified in the PRINT command, the second line number will appear on the fourth line of the listing.) Line numbers start at one and ascend consecutively to the end of the file. A line number may not exceed 99999.

P=number sets the length of the page. Each page will consist of the indicated number of lines, where "number" is a decimal number between 3 and 255. The default page length is 66 lines.

T=number sets the line number of the beginning of the text to a decimal number between 1 and 255. The default value for this argument is 2.

B=number sets the line number of the end of the text to a decimal number between 1 and 255. The default value for this argument is 63. The number must be at least 2 less than that specified in the P argument.

L=number determines the character position of the first character of each line on the listing. "number" must be a decimal number between 1 and 255. The default value for this argument is 1.

R=number determines the character position of the last character of each line on the listing. "number" is a decimal number between 1 and 255; its default value is 64.

If "number" is not specified or is Ø in any of the foregoing arguments, the default value is assigned to that argument.

NOTES:

The output file for this command need not be a device driver. For example, to change the format of a long text file so that it is double-spaced, you might:

1) Print the file to a text file, setting the D flag in the PRINT command line.

2) Read the resulting text file into EDT3, which automatically deletes linefeeds.

3) Write the text file from EDT3.

The text file resulting from this procedure may be edited thereafter in EDIT.

EXAMPLES:

*PRINT R=8Ø,O=TEXT,S=AP,FILE1

Copy the ALS-8 file FILE1 to the file TEXT. Output will be paginated and will consist of 8Ø-character lines.

*PRINT S=D#,FILE1,H="FILE2",S=-#P,O=#5,FILE2

Copy FILE1 to the CI output file; the listing will be double-spaced and contain line numbers. Then copy FILE2 to the open file whose number is 5; the listing will be paginated, double-spaced, and have the string FILE2 as a heading at the top of every page.

RANDOM - Create index block for file

RANDOM file{,}

OPERATION: This command causes an index to be built for the specified file on the specified unit; as a result, the file is made randomly accessible, i.e., it is possible to SEEK to a particular byte of the file without proceeding sequentially through all of the bytes before it. It is not possible to perform this operation on a device file.

ARGUMENTS:

file — is the name or number of the file to be made random.

* — means that the file should be left open at the conclusion of the RANDOM operation. The number of the open file is displayed on the CI output file, and the cursor is left at the end of the file.

NOTES: Detailed material about file access can be found in Section 5 of this manual.

EXAMPLES:

```
*RANDOM #3
*RANDOM POTTS
```

READ - Transfer contents of file to memory

READ file{,addr1{,addr2 or >count}}{,}

SAFE

OPERATION: Data is read from the specified file, which must exist, to a specified or default address. Reading progresses until:

1) the end of the file is reached, or

2) data has been read into memory addresses up to and including a specified last address, or

3) a given number of bytes has been read.

The number of bytes read (the "LOAD COUNT") is displayed on the CI output file.

ARGUMENTS:

file — is the name or number of the file from which the data is to be read.

addr1 — is the lowest of the consecutive addresses into which the data will be loaded. The default for addr1 is 256 (100H).

addr2 — is the highest address into which data may be loaded.

>count — is the number of bytes to be read. Do not specify both addr2 and >count.

* dictates that the input file be left open at the conclusion of the READ operation. The number of the file is displayed on the CI output file.

NOTES:

READing an image file, instead of loading it, is a way of localizing all of its data (including headers), whereas if the file were loaded, individual image blocks might be scattered all over memory. You might want all of the data to be contiguous in memory so that you can 1) find out the number of bytes in the file, or 2) use DEBUG to examine the program without having to write over data stored at locations your program would normally occupy. (If you READ a program, instead of loading it, it cannot be executed from memory.)

If you want to READ a device file as it appears on the diskette, change its type to Image before you READ it.

EXAMPLES:

*READ FILE1

Read all of FILE1 to addresses starting at 100H.

READ FILE1,0,>1000,

Read the first 1000 bytes of FILE1 to addresses starting at 0. If there are fewer than 1000 bytes in the file, reading will stop when the end of the file is reached. FILE1 will be left open at the conclusion of the operation.

*READ FILE1,100,9000

Read FILE1 to addresses starting at 100H. Reading will stop if the end of the file is reached, or when data has been loaded to address 9000H.

REATR - Change protection attributes of file

*REATR filename {,new attributes}

SAFE

OPERATION: The protection attributes of the file are changed in accordance with the second argument in the command line. (Protection attributes are discussed in Section 5 of this manual.)

ARGUMENTS:

filename is the name of the file whose attributes are to be changed.

new attributes determines what changes will be made in the attributes of the file; it consists of from Ø to 8 attribute codes, each preceded by a plus (+), a minus (-), or no prefix at all. The codes are:

K - Kill protect (no KILL)
W - Write protect (no WRITE or ENDF)
R - Read protect (no READ)
I - Information protect (no GET, SAVE, or FILES without special arguments)
A - Attribute change protect (no REATR)
N - Name and type change protect (no RENAME or RETYPE)
E - Disk allocation protect (This attribute makes it impossible to lengthen or shorten the file.)

If no prefixes appear in the string of codes, the new attributes simply replace the old ones.

If prefixes do appear, a plus (+) means that the following attribute should be ADDED to the list of old attributes, and a minus (-) means that the following attribute should be DELETED from that list. As long as there is at least one prefix in the string, a code without a prefix is treated as though it were preceded by a plus (+).

Attribute codes may be listed in any order. If no new attributes are supplied, all existing attributes are removed from the file.

EXAMPLES:

*REATR FILE1,KWRIANE

Turn on all protection. A file with these attributes is useless, because you cannot write to it, read from it, kill it, or change its attributes.

*REATR FILE1,KW

Protect FILE1 agains any operation that would kill it or write to it, and remove all other attributes, if there were any.

*REATR FILE1,+KW-R

Add kill and write protection to the current attributes of the file, and remove read protection. Leave other attributes unchanged, if there are any.

RECOVER - Reclaim lost space on diskette

*RECOVER {/u}

OPERATION: RECOVER searches the directory on the specified unit and finds all File ID's associated with known files: that is, with files that have entries in the directory. (Each file on a diskette has a unique file ID, not to be confused with a file number. File ID's are used for system book-keeping; you will never need to refer to a file by its ID.) Then each sector of the diskette is read and handled as follows:

If the sector cannot be read, it is rewritten with useless data. The data it once contained is lost.

If the sector is readable but has a file ID of Ø, or some other file ID not associated with a known file (see above), the sector is rewritten with useless data. For example, if you KILL a file and get a KILL error, the sectors following the error will fall into this category, because they will no longer be associated with a file known to the directory.

The free space map (FSMAP) is rewritten to show that the reclaimed sectors are available for use. (The free space map is described in Section 5 of this manual.)

ARGUMENTS:

/u stands for the unit that contains the diskette from which data is to be recovered. If this argument is absent, the default unit is used.

NOTES:

This program will not recover data from a file that has been partially destroyed. Try to recover all available data by using more primitive commands like READ and SPACE. If you have a backup copy of the bad file and therefore do not need to recover the data, KILL the file before you use RECOVER; otherwise the sectors after the error will not be reclaimed, because their file ID will still be known to the directory on the diskette. Never use RECOVER unless you have made a DISKCOPY of the bad diskette.

RENAME - Change name of file(s)

*RENAME oldname,newname{,oldname,newname...}

SAFE

OPERATION: The name of the specified file is changed to the specified new name. The renamed file is not altered in any other way.

ARGUMENTS:

oldname is the current name of the file to be renamed.

newname is the new name to be given to the file.

Both arguments must be legal PTDOS filenames. oldname may have a unit number appended to it; newname never needs a unit number, because the new name will always be given to the old file, on the same unit on which that file presently exists.

NOTES:

If a file to be renamed has a name change protection attribute (the N attribute), an error message will be printed.

If a file with the new name already exists on the diskette, the program will be aborted and an error message will be generated.

EXAMPLES:

```
*RENAME GOOP,POTTS
*RENAME OLD1,NEW1,OLD2/1,NEW2
```

RETYPE - Change the type of a file

*RETYPE filename,newtype

SAFE

OPERATION: The type of the named file is changed to the specified new type. If the file is protected against a change in its type, it must be given new attributes before its type is changed.

ARGUMENTS:

filename is the name of the file whose type is to be changed.

newtype is the type to be assigned to the file.

There is a discussion of file types in Section 5 of this manual. If the desired type is a printable character, enter that character as the second argument in the RETYPE command line. To specify a type with a value up to 20H, enter the corresponding control character. Specify an image type file as type 'I' or as 'I' followed by another character. A non-image type may NOT consist of more than one character. If the file contains a device driver, its type should be 'D'.

NOTES:

There are several reasons you might want to change the type of a file. One reason is that some operations may not be performed on files of a certain type; changing the type of the file may make it possible to perform the desired operation. Another reason is that types are a convenient way of grouping files, and your groupings may change.

EXAMPLES:

```
*RETYPE RECORDS,A
*RETYPE PROG1,IS
```

RNUM - Renumber lines of text file

*RNUM filename{<A>}{,number}{,I}

OPERATION: This command replaces the first four characters of consecutive lines of a text file with consecutive line numbers starting at "number" and ascending in increments of 1 until the end of the file is reached or the line number exceeds 9999. Renumbering may be desired if some lines of a program file have been deleted or rearranged in EDIT and you want to restore a sequential numbering scheme to the file.

ARGUMENTS:

filename is the name of the file that will receive line numbers in character positions 1-4 of each line.

<A> means that the file is in ALS-8 format.

number is an optional starting number for the numbering sequence. The default value for this argument is 1.

I means that COPY files will not be renumbered. If this argument is absent, COPY files other than the file called PTDEFS will be renumbered with the main source file. (In this context, "COPY" refers not to the COPY command, but to the COPY pseudo-operation of the assembler.)

NOTES:

Remember that if the named file that does not contain line numbers, RNUM will replace the first four characters of each line of the file with a line number. Thus, the line

The cat is on the mat

will become

000lcat is on the mat

SAVE - Write one or more files to an archive file

*SAVE O=file{,/u}{,T=type}{,strings}{,S={-L}{-I}}

OPERATION: All files satisfying the specified conditions are written onto the file indicated as the output file. A SAVE output file is an archive file that contains other files. It is possible to retrieve files from an archive file by using the GET command.

ARGUMENTS:

O=file specifies the archive file by its name or number. (That file may be a device file.) File names and other information are written to the output file along with the data so that SAVE files may later be retrieved by name with the GET command.

/u the unit to search for the files to be SAVEd. If this argument is absent, the default unit is searched.

T=type dictates that only files of the specified type will be SAVEd.

strings have the same significance as in the FILES command, above.

S=options

- -L causes the listing of the names of SAVEd files to be suppressed.
- -I makes it possible for information-protected files to be SAVEd.

NOTES:

When a file is made part of an archive file, it does not have an individual directory entry. If the file also exists independently on the diskette, the independent file will have its own directory entry.

EXAMPLES:

*SAVE /1,O=CTAPE1,T=S

All files of type S on unit 1 are written onto a file called CTAPE1.

*SAVE O=BACKUP,<PAY>,<OLL

All files whose names contain the letters PAY or that end with OLL are written onto a file called BACKUP.

SEEK - Position the file cursor

*SEEK file,number{,B}

SAFE

OPERATION: This command moves the cursor of the named file to the specified byte or to the first byte of the specified block of the file. This operation may be performed only on a "random" file. This command differs from SPACE in that SEEK moves the cursor to an absolute position in the file, whereas SPACE moves the cursor to a position relative to its current position. PTDOS manages buffering, so that the cursor may be moved beyond the limits of the current block.

ARGUMENTS:

file — is the name or number of the file whose cursor is to be moved. This file is always left open after the cursor has been moved; the number of the open file is displayed on the CI output file.

number — is the number of the byte or block at which the cursor should be positioned. "number" is assumed to be hexadecimal, unless it is followed by a colon and a letter code (see the introduction to this section).

B — means that the cursor should be positioned at the first byte of block "number" of the file. If this argument is absent, "number" is assumed to point to a byte, rather than to a block.

NOTES: To make a file randomly accessible, use the RANDOM command.

EXAMPLES:

*SEEK POTTS,125
*SEEK #3,2F,B

SET - Change system parameters in memory

*SET argument{,argument....}

SAFE

OPERATION: This command sets the values of various system parameters. SET differs from CONFIGR in most of the parameters altered by SET are altered only in memory, whereas CONFIGR alters them on the diskette but not in memory.

ARGUMENTS: Arguments may appear in any order; each has the form "argument=value." In the following discussion "number" is always a decimal number, and "address" is always a hexadecimal number. These are the possible arguments:

DU=number determines which unit will be regarded as the default unit. For example, if DU=1 and you enter a filename without a unit number, the file is presumed to reside on unit 1. "number" must be less than 8 and should not exceed the number of available units minus 1.

EF=number sets the number of the "echo file" on which command lines will be duplicated when the SW=E switch is on (see below). Output from the commands will not go to this file.

SY=number sets the number of permanently open files. "number" cannot be greater than 7.

PR=address sets the lowest memory location into which programs or data can be loaded. For example, if PR=100, no programs or data will be loaded at any lower address.

NU=number sets the number of null characters that will be sent to the console after every carriage return. Set this parameter if your output device requires a delay between lines; otherwise, the value is 0 and there is no need to change it.

BU=number sets the lowest address of the system buffer area. The default value is 9000H; change it if you intend to open files whose combined block sizes exceed the available buffer area.

WC=address points to the starting address of the write character routine for the console driver. (See Section 9.4.3.)

RC=address points to the starting address of the read character routine for the console driver. (See Section 9.4.3.)

SC=address points to the starting address of the test for waiting character routine for the console driver. (See Section 9.4.3.)

DA=mm.dd.yy sets the date in the System Global Area. Slashes (/) may be used instead of periods.

DD=mm.dd.yy sets the date on the default unit disk. Slashes (/) may be used instead of periods. THIS VALUE IS WRITTEN TO THE DISKETTE; all other arguments alter memory only.

NA=string sets the disk name in the System Global Area. This name will appear in the heading of output from the SYST / command.

SW=switches sets one or more "switches" in the System Global Area. If a switch is on, the option it names is in effect; if it is off, the option is not in effect. The letter codes for the switches may be in any order, and should not be separated by commas. A '+' or no prefix before a letter code turns the related switch on; a '-' turns it off.

CODE	FUNCTION IF SWITCH IS ON
E	Echo commands to the CI echo file (see EF=, above).
L	Lock the system so that nothing can be written to any file on any unit.
U	Upshift all ASCII characters input from or output to the console, i.e., make all letters upper case.
H	Operate in "verbose mode."
V	Verify every byte written (by reading it after writing it).
B	Operate in binary mode (all 8 bits, no upshift, no extra NULs after LF).

EXAMPLES:

```
*SET DA=Ø6.28.77,DU=1,SW=E
*SET SW=-ELU,PR=255:D
```

SETIN - Make named file the CI input file

```
*SETIN file{,*}
```

SAFE

OPERATION: The current input file is closed (unless it is file #Ø), and the named file becomes the CI input file; that is, the next command read by the Command Interpreter (see Section 3) will be read

from that file. If the named file is already open, the cursor remains at its current position in that file. When an end-of-file is encountered on the input file, file #Ø again becomes the CI input file. The named file need not be a device file (see NOTES).

ARGUMENTS:

file	is the name or number of the new CI input file.
*	means that the previous CI input file should be left open (although the Command Interpreter will not receive input from it). Otherwise that file is closed, unless it is one of the permanently open files, in which case it is left open but not used for input.

NOTES:

This command allows a frequently executed group of commands to be stored in a file and executed as a "job." Develop a text file consisting of the series of commands, just as you would enter them from the keyboard (remember that there will be no prompt). Then use the SETIN command to make that file the CI input file.

SETOUT - Make named file the CI output file

SETOUT file{,}

SAFE

OPERATION: The named file becomes the CI output file; that is, all output will go to that file until another file is made the output file. The CI output file need not be a device file.

ARGUMENTS:

file	is the name or number of the new CI output file.
*	means that the previous CI output file should be left open (but receive no output). Otherwise, that file is closed, unless it is one of the permanently open files, in which case it is left open but receives no output.

EXAMPLES:

```
*SETOUT #1
*SETOUT POTTS,*
```

SPACE - Move the file cursor

*SPACE file,how

SAFE

OPERATION: This command moves the cursor of the named file to the beginning or end of that file, or a specified number of bytes forward or backward. SPACE differs from SEEK, in that SPACE moves the cursor to a position relative to its current position, whereas SEEK moves the cursor to an absolute position in the file. It is possible to SPACE through a random access file, as well as through a sequential access file. PTDOS manages buffering, so that the cursor may be moved beyond the limits of the current block.

ARGUMENTS:

file — is the name or number of the file whose cursor is to be moved.

how — indicates the desired direction and extent of motion:

>count	moves the cursor forward "count" bytes. "count" is a number.
<count	moves the cursor backward "count" bytes. "count" is a number.
<	moves the cursor to the beginning of the file.
>	moves the cursor to the end of the file.

* — means to leave the file open after the cursor has been moved. This argument is really superfluous, because the file is left open, anyway. The number of the open file is displayed on the CI output file.

NOTES:

If the end or the beginning of the file is reached, making it impossible to SPACE the desired number of bytes, a message is displayed to indicate how far (in bytes) the cursor was actually moved, and whether the cursor is now positioned at the beginning or the end of the file.

EXAMPLES:

```
*SPACE FILE3,>500     (forward 500 bytes)
*SPACE FILE3,<        (rewind the file)
```

SYST - Display system parameters

*SYST {L}
*SYST /{u}{,L}

INTERRUPTABLE

OPERATION: This command displays the values of certain parameters in the System Global Area (see Subsection 6.3). The first form of the command causes information to be taken from the System Global Area currently in memory; the second form causes information to be taken from the System Global Area File on the specified or default unit. (The default unit is defined in the System Global Area.)

ARGUMENTS:

u is the unit from which information is to be taken.

L dictates that more information be displayed than if this argument were absent.

NOTES:

If L is not present (in either form of the command), the following information will be presented:

The PTDOS version number (and the unit number, in the second form of the command)
The disk name, and the date in the System Global Area
The lowest address assigned to the system (GLLOW)
The amount of unused buffer space, in hexadecimal and decimal

If the L is present, the following items of information are added:

The user memory protect address (GLPRO)
The status of the disk write lock, on or off (GLLOK)
The number of the default unit (GLUNI)
The number of permanently open files (GLPRM)
The file number of the current utility file (GLUTF)
The number of the CI input file (GLCIF)
The number of the CI output file (GLCOF)
The number of the CI echo file (GLCEF)
The maximum number of disk drive units allowed (GLMXU)
The starting address of the console input routine (GLRCH)
The starting address of the console output routine (GLWCH)
The starting address of the console test for waiting character routine (GLTCH)
The number of nulls to be sent after a linefeed (GLNUL)
The status of the system log file, on or off (GLLOG)

After this information is displayed, hit any key to return to PTDOS command mode.

TREK8Ø - A Video Star Trek Game

*TREK8Ø

OPERATION: Loads and executes the TREK8Ø game program. See the TREK8Ø User's Manual, in a separate volume, and ignore all information pertaining to loading the program from cassette.

WRITE - Write contents of memory to a file

WRITE file,{!blksize,}addr1,addr2 or >count{,}{,<}

SAFE

OPERATION: This command writes the contents of a specified area of memory to the named file. It is possible to specify the lowest address whose contents are to be written, as well as either the highest address or the number of bytes to be written to the file.

ARGUMENTS:

file	is the name or number of the file to which data is to be written. If "file" is a name and the file does not exist, it is created with type '.' and the specified or default block size.
!blksize	is the block size to be assigned to the named file if that file must be created. If this argument is absent, the file will be created with block size 4CØH. If the file already exists, this argument will be ignored. The ! is literal.
addr1	is the first address from which data is to be written. The default value for addr1 is 1ØØH.
addr2	is the last address from which data is to be written. Either this argument or >count must be supplied.
>count	is the number of bytes to be written.
*	dictates that the output file be left open after it is written. The number of that file is displayed on the CI output file.
<	dictates that an end-of-file be written to the output file at the conclusion of the WRITE operation.

EXAMPLES:

```
*WRITE FILE2,100,>20        (20H bytes, starting at 100H)
*WRITE FILE2,7FF,4580,<     (end of file written to output file)
```

XREF - Generate cross-reference listing of assembly language file

*XREF infile,outfile{,S=options}{,top of memory}

INTERRUPTABLE

OPERATION: This command generates a cross-reference listing of the input file (which must be an assembly language source program) and writes that listing to the output file. XREF produces the listing much faster than the assembler does. Each line of the cross-reference listing consists of a label defined in the program, the value of that label in hexadecimal, the number of the line at which the label is defined in the source program, and the numbers of all lines that contain references to the label. The value and the line number are separated by a slash; all other items are separated by blanks. COPY files will be processed with the main source program. Undefined labels have a value of FFFF and a defining line number of ????.

ARGUMENTS:

infile — is the name or number of an assembly language source file.

outfile — is the name or number of the file that will receive the listing. This file may be an output device or a file on the diskette.

S=options — if present, MUST be the argument after "outfile."

- D causes the source program to be displayed on the console output device while the cross-reference listing is being generated and sent to the output file.
- V causes the output to be formatted for the video display, i.e., to be printed with 64 characters per line. Otherwise, output width is 72 characters.
- # causes line numbers to be assigned to the source code file, in place of those which exist there. If the source code file does not have line numbers, numbers will be added by default, even if this option is not specified.

R dictates that the CPU registers be included as labels in the cross-reference listing.

P instructs XREF to paginate output to the listing file.

1 or 2 or 3 specifies page width
1 - 72 columns
2 - 80 columns (default)
3 - 132 columns

top of memory is the highest address that may be used by XREF. A symbol table is created immediately after the XREF code. A cross reference table is constructed downward from the lowest of: 1) the specified top of memory, 2) the last available memory location, and 3) the bottom of the PTDOS buffer area.

If these tables collide, a disk overflow file (XREFOVER) will be created and XREF will continue to operate, using the disk file instead of memory.

If the symbol table should overflow available memory the message:

SYMBOL TABLE OVERFLOW

will be displayed.

If the reference table overflows the disk file (XREFOVER), the message:

REFERENCE TABLE OVERFLOW

will be displayed. In either case XREF will terminate.

EXAMPLES:

```
*XREF SOURCE,CROSS/1,S=R,8000
```

Generates a cross-reference listing of SOURCE and writes it to CROSS on unit 1, with CPU registers included as labels. XREF will not use memory between 8000H and the lowest address assigned to PTDOS.

ZIP - Fill memory with number

*ZIP {number}

OPERATION: All memory locations below GLLOW (the lowest system address) are set to the specified value.

ARGUMENTS:

number — is the value to be placed in every memory location between Ø (or a user memory protect) and the lowest system address. The default value for this argument is 76H (the 8Ø8Ø HALT instruction).

EXAMPLE:

*ZIP FF

2.3 COMMANDS NOT USUALLY ENTERED FROM THE KEYBOARD

PTDOS provides a group of commands used most frequently in DO files (see Section 4) and in text files that are used for CI input (see the description of SETIN, above). With the exception of $STOP, the commands in the list below may be entered from the keyboard. The $ sign that precedes each command is literal.

ARGUMENTS:

$LST — turns on the CI echo flag, so that subsequent commands will be echoed to the CI echo file or to the system log file, if there is one. (Same as SET SW=E.)

$NLST — turns off the CI echo flag. (Same as SET SW=-E.)

$REM string — identifies the string as a remark, so that the Command Interpreter will not interpret it, or any part of it, as a command. The string may include commas; it may not, however, include a semi-colon or a carriage return, because either of these two characters is regarded as a terminator. If the echo flag is on, the entire $REM command line will be echoed.

$PR string — causes the string to be printed on the CI output file. Control characters may be included in the string.

$WAIT — waits for a carriage return or a MODE SELECT (CTRL-@) to be entered from the keyboard. If a carriage return is entered, execution proceeds to the next command. If a MODE or CTRL-@ is entered, all files are closed, except the permanently open files, and the console becomes the CI input file. If the carriage return or MODE was entered during the execution of the last command, and if that command was not "INTERRUPTABLE," the $WAIT command reads the <cr> or MODE and proceeds accordingly.

$ESC determines whether a MODE SELECT (or CTRL-@) has been entered from the keyboard. If so, the keyboard becomes the CI input file; if not, execution proceeds to the next command. Because $ESC, unlike $WAIT, does not wait for an entry, a MODE SELECT will be perceived only if it was entered during the execution of the last command, and only if that command was not "INTERRUPTABLE."

$STOP is the command used to return to the system from a DO file. Do not use the command in any other context.

$CREATE is identical to CREATE, except that:

1) if the file already exists, no error is reported (in fact, nothing happens), and

2) a list of attributes may be included as a fourth argument on the command line. Attributes are specified just as in the REATR command. For example,

   $CREATE NAME,T,4CO,KIN

   will be kill-protected, information-protected, and name-and-type change-protected.

SECTION 3

COMMAND INTERPRETER

3.1 INTRODUCTION

The Command Interpreter (CI) provides the fundamental interface between the user and PTDOS. It accepts commands from the system console or a command file and executes them one by one. Whenever the CI is waiting for input, it sends an asterisk (*) prompt to its output file (usually the console).

A PTDOS command has two parts: a program (image file) name and an optional argument list. For example, in the command FILES S=-I, which displays a list of all files on the default diskette, FILES is the name of an image file that contains the program to produce such listings, while S=-I is an argument instructing FILES to list all files, whether or not they are information protected.

The Command Interpreter reads the name of the image file, loads that file into memory, and usually causes it to begin execution. The program is then responsible for reading and interpreting its own arguments, if it has any.

3.2 COMMAND SYNTAX

A command line has the following syntax:

program {{,}{program}...}{_arguments}{;command...} <cr>

The line may be no longer than 80 characters. Notice that the program name is separated from its arguments by one or more blanks, shown above as an underline (_) for clarity. Two or more complete commands may be placed in the same command line if they are separated by semi-colons (;). The last command in the line is terminated by a carriage return.

Program names appearing in a command line must be image files, or the CI will reject the command and generate an error message. The CI can only start the execution of a program if the image file has a starting address. Image files and starting addresses are discussed in Section 5.

If a program name is followed immediately by a comma (,) it will be loaded but not executed, even if it contains a starting address. If a command line consists of several program names separated by commas, each of the programs will be loaded at its load address, and the last will be executed unless it is followed by a comma or has no starting address. Any arguments that follow such a list of program names must be read and interpreted by the last program in the list.

Since arguments are read by individual programs rather than by the CI, the system places no restrictions on their format, except that a semi-colon(;) cannot occur within an argument list. However, all programs supplied with PTDOS use the same argument syntax, which is recommended for user programs as well:

1) Blanks in argument lists are ignored.

2) Arguments are separated by commas.

3) Arguments may be either positional or keyword-delimited. Positional arguments must occur in a prescribed order; if any are to be omitted from the middle of a list, an extra comma must be used to hold the place of each one missing. Keyword-delimited arguments have the form keyword = value (e.g., S=-I). They need not occur at a particular place in an argument list, and can even be placed between two positional arguments or omitted altogether without requiring a placeholder.

3.3 OPERATION

3.3.1 CI Files

The Command Interpreter gets its input from and sends its output to files defined by two global area parameters:

GLCIF	CI input file number
GLCOF	CI output number file

These parameters are set initially to the system console, files #Ø and #1. They may be altered by the SETIN and SETOUT commands so that the CI executes commands or sends its output to another file.

Once it gains control, a program reads its arguments from the current CI input file; it must obtain the number of this file from the system global parameter GLCIF. If a program with optional arguments in fact has none, the first character it reads from the CI input file will be the carriage return or semi-colon (;) that terminated the command.

The CI input and/or output files may be disk files or device files, for example, a special terminal device. The driver for a device to be used as the CI input file must not be read protected, and must support the following operations:

DTRB	read block;
DTRNB	read next block;
DTRLB	read last block.

The last operation (DTRLB) is required only if commands cross block boundaries and are being echoed (see below). The driver for a device to be used as the CI output file must not be write protected, and must support the following operations:

DTWB	write block;
DTWBR	write block, read next block.

If an end of file is encountered while a program name is being read from the CI input file, the following actions are taken:

1) The command is ignored.

2) If the CI input file was the system console, all open files are closed. An EOF on the system console is generated by typing a CTRL-C.

3) If the CI input file was not the system console, it is reset to the system console. No open files are closed, including the original CI input file.

The CI may echo commands to a selected file. Echoing is controlled by three system global parameters:

GLECF	echo file number;
GLECH	echo enable switch;
GLLOG	log file enable switch.

Commands will be echoed to the file whose number is in GLECF if GLECH is non-zero. GLECF has the default value of 1 if GLLOG was zero when the system was bootstrapped. Then, if echoing is enabled, all commands will be echoed on the system console. It is often useful to monitor the command stream on the console if CI input is coming from a disk file. If GLLOG was non-zero when the system was bootstrapped, the system log file SYST.LOG will be permanently opened as file #3 and GLECF will be set to 3, so that all commands will be echoed on SYST.LOG. The system log file is intended to preserve a record of all commands executed, to facilitate later auditing and error recovery.

If echoing is enabled, the CI must read an entire command, including arguments, from its input file. To make the arguments available to the program for which they are intended, the CI backspaces its input file to the beginning of the arguments after the command has been echoed.

3.3.2 Invocation

The Command Interpreter may be invoked in one of two ways: a "reset" (initial system start-up, RESET or ABORT system calls, or the SRESET system entry point), or a "return" caused by a normal return to the system (RETURN or RETURN AND SET TRAP system calls).

On either a reset or a return, the following actions are taken:

1) The stack is switched to the internal CI stack.

2) The error trap addresses are set as follows:
GLERH=Ø, GLERM=Ø, and GLERS=-1.

3) The interrupt routine addresses are set to disable interrupt handling: GLBD1=-1, and GLBD2=-1.

On a reset, the following additional actions are taken:

1) The CI input and output files and the current utility file are reset to their standard values: GLCIF=Ø, GLCOF=1, and GLUTF=2.

2) The buffer for the CI input file (#Ø) is cleared so that no partial or complete commands will be executed when the CI gains control.

3) The file #Ø prompt is set to an asterisk (*).

4) The CI return trap is disabled: GLTRP=-1.

5) Control passes to the CI.

On a return, the CI return trap address is examined to see whether control should pass to the Command Interpreter or to a user routine. If GLTRP=-1, the CI gains control. Otherwise, control passes to the routine whose starting address is in GLTRP.

SECTION 4

COMMAND MACRO PREPROCESSOR

4.1 INTRODUCTION

In the discussion of the SETIN command in Section 2, it is mentioned that a series of frequently executed commands may be put in a standard text file and later executed with the command:

```
*SETIN file
```

where file is the name or number of the text file containing the commands (see Section 2). The text file consists of commands whose syntax and punctuation (including semi-colons and terminal carriage returns) are EXACTLY the same as if the commands were being entered from the keyboard. For example, if a file named FIRSTJOB contained the statements:

```
CREATE FIRST,T,4CØ<cr>
EDIT FIRST<cr>
```

(neither carriage return is literal), the command SETIN FIRSTJOB would cause the Command Interpreter to read the two commands from that file and execute them, just as though they had been typed from the keyboard.

The DO program on the PTDOS diskette enables the user to create macro command files more versatile than the FIRSTJOB type, because it allows variables to be inserted in a file to represent parameters that the user will supply in the command to execute the macro. The DO program also provides for conditional substitutions based on the presence or absence of expected parameters in the DO command line.

The remainder of this section is devoted to prescribing the following procedure for using the macropreprocessor.

1) Using EDIT or EDT3, develop a text file containing commands and either variables or actual parameters. (The rules for specifying parameters will be described below.)

2) Give the DO command, specifying the name of the text file and the actual parameters to replace the variables in the text file. The program will create an output file in which the variables have been replaced by the actual parameters in the DO command line. The DO command can also take arguments which select the name of the output file (otherwise a default name is used) or suppress execution of the output file after preprocessing.

3) If automatic execution of the output file has not been suppressed, that file is made the CI output file; if execution has been suppressed, the output file may be executed at any time with the SETIN command. (The input file, unless it contains no dummy parameters, cannot be executed successfully with SETIN.)

At the conclusion of this procedure, there are at least three new files on the diskette: 1) the input text file, which can again be executed with the DO command, 2) at least one output file, which can be executed with SETIN, and 3) a stack file called $DO.STK$, which DO uses to implement recursion (see below) and to provide for an orderly return to the system when a macro is terminated. (There will be more output files if the input file includes one or more DO commands.) If you want to be able to execute the same set of commands repeatedly, without changing any parameters, save the first output file and thereafter use SETIN; if you want to execute the commands, but not always with the same parameters, save the input file and thereafter use DO.

4.2 DEVELOPING THE INPUT FILE

An input file is a normal text file consisting of commands. Any PTDOS image file may be used in a macro; if the image file is not the last to be executed, it should end with a RETOP (return to the operating system), because otherwise the subsequent command(s) in the macro file will not be executed. The system commands that begin with the dollar sign ($) are more frequently used in macros than in any other context.

It is permissible for a macro to execute itself recursively, for example, for a file named AGAIN to include a DO command to process and execute the file AGAIN. It is also possible for a macro to execute another macro. Whenever a macro is executed, even if it is being executed recursively, another output file is opened; DO is able to generate at least 36 such files, as long as PTDOS is configured to allow adequate buffer space and the required number of open files.

The last command in the input file should be $STOP.

4.2.1 Variables

A macro input file may contain as many as 36 variables, each designated by a percent sign (%) followed by a character in the range Ø-9,A-Z. (Upper and lower case letters are equivalent in the representation of a variable.) Each of these variables represents a parameter passed to the program in the DO command line: %Ø represents the name of the macro input file, %1 represents the first subsequent parameter on the line, %A represents the tenth subsequent parameter, etc. For example, if a file named MAKE consists of the command lines:

```
CREATE %1,T,4CØ;EDIT %1<cr>
$STOP<cr>
```

the command DO MAKE,JACKAL will produce an output file that creates a file called JACKAL, with dynamic buffering and a block size of 4CØ, and then enter the editor to edit the newly created JACKAL file. (Notice that the same variable may occur more than once in the input file.) If MAKE consisted of the command lines:

```
CREATE JACKAL,%1,%2,<cr>
%3 JACKAL<cr>
$STOP<cr>
```

the command DO MAKE,T,4CØ,EDIT would have exactly the same results.

If a variable is specified and there is no corresponding parameter supplied on the DO command line, an error will be reported.

4.2.2 Conditionals and Signed Conditionals

DO provides for two kinds of conditionals. The simplest kind of conditional has the form [string], where brackets are literal and "string" is either a variable or a string containing a variable. If the parameter corresponding to the named variable is actually supplied on the DO command line, "string" is written to the output file, with the variable assigned its actual value; if the string contains only a variable, the value of the variable is written to the output file. If the parameter corresponding to the named variable is absent, "string" is not written to the output file, and processing of the input file continues. The brackets will never appear in the output file.

EXAMPLES:

1) If MAKE consists of the line:

```
CREATE %1,[%2],[%3];EDIT %1;$STOP<cr>
```

the command DO MAKE,POTTS,,24Ø will produce a command file:

```
CREATE POTTS,,24Ø;EDIT POTTS<cr>
```

whereas the command DO MAKE,POTTS will produce:

```
CREATE POTTS,,;EDIT POTTS<cr>
```

In neither case will an error be reported, as it would be if the variable names were not enclosed in brackets. Also notice that the commas in MAKE are outside the conditional; it is important to provide for commas in the case of any command whose arguments are positional, i.e., to be entered in a specific order. An extra comma is required in the command DO MAKE,POTTS,,24Ø to signify that the second parameter has, in fact, been omitted from the parameter list; otherwise, the value 24Ø would be substituted for %2.

2) If a file named EX5$ contains the command lines:

```
$REM [%1 IS PRESENT.[SO IS %2.][ AND ALSO %3]]<cr>
$STOP<cr>
```

the command DO EX5$,,arg1,arg2 will produce:

```
$REM<cr>
```

whereas the command DO EX5$,arg1,,arg3 will produce:

```
$REM arg1 IS PRESENT. AND ALSO arg3
```

This example contains "nested" conditionals. If the parameter corresponding to the variable in the outermost conditional is not present, the string, including any other conditionals, will not be written to the output file. Conditionals may be nested to a depth of 14; the example above is nested to a depth of 1.

The second kind of conditional is a "signed" conditional of the form [string] where "string" consists of a percent sign, a plus or minus sign, a variable designation, and a string. Signed variables are used only in conditional expressions; they have the meaning "write the string to the output file ONLY if the parameter corresponding to the named variable is present (for a plus) or absent (for a minus)." The actual parameter is never written to the output file in the case of a signed conditional.

EXAMPLES:

1) The command line:

```
CREATE %1,[%2][%-2T],[%3][%-3100]
```

results in the substitution of the appropriate values for %2 and %3, if those parameters are present in the DO command line. If the second parameter is absent, the value T will be written to the output file; if the third parameter is absent, the value 100 will be written to the output file. Obviously, only one of each pair of conditionals can be true.

2) The conditional expression:

```
[$PR %-1%+2%+3-4 OK]
```

produces $PR OK only if parameters %1 and %4 are absent and parameters %2 and %3 are present.

This kind of conditional can also be nested to a depth of 14.

4.2.3 Cases of Special Punctuation

1) If a line of the input file must overflow to the succeeding line (for example, if the input file is being prepared in EDIT

and is more than 64 characters long), type a backslash (\) after the last character that will fit on the first line; the backslash will prevent the carriage return that follows it from being copied to the output file. For example,

```
CREATE %1\<cr>
;EDIT %1<cr>
```

will produce CREATE %1;EDIT %1<cr>. As the example illustrates, the backslash may be used even if the first line really would not overflow to a second line. By this means a line of a macro may be extended to a length of 255.

2) A backslash must also be used to indicate that a bracket ([or]), a percent sign (%), or a backslash (\) should NOT be regarded as a directive to the program, but should be copied to the output file "as is." The backslash must occur immediately before the character to be copied. The character to be copied may be another backslash; in that case, the second backslash has no effect on the character after it.

EXAMPLE:

If a file named GROCLIST contains the line:

```
$REM [%1 FOOD]<cr>
```

the command DO GROCLIST,DOG will produce:

```
$REM DOG FOOD
```

If the line is:

```
$REM \[\%1 FOOD\] = \[%1 FOOD\]
```

the same command will produce

```
[%1 FOOD] = [DOG FOOD]
```

3) If a space ( ), a comma (,) or a single or double quote (' or ") is to be included in an parameter and written to the output file, that argument must be surrounded by either single or double quotation marks. For example, the command line:

```
DO STREET$,"1234 8th ST., BERKELEY, CALIFORNIA"
```

will cause the whole address, not including the quotation marks, to be substituted for variable %1 and written to the output file.

4) If an argument is to be present, but contain no characters, it should be represent by adjacent single or double quotation marks.

4.3 EXECUTING DO

Many of the examples above illustrate the standard syntax of the DO command:

```
*DO infile,parameter,parameter,parameter....
```

There are a few arguments, which may be inserted in any order BEFORE THE NAME OF THE MACRO INPUT FILE.

O=file name

is used to specify the name of the output file to be written by the DO program. If this argument is absent, the file is called $DO.CMDx, where x is a digit. The first file opened is $DO.CMDØ; if additional output files are required (because of recursion or nesting of DO's), those files will be named $DO.CMDl, $DO.CMD2, etc. As many as 36 output files may be opened, if the configuration of PTDOS presents no other restrictions. If this argument is present, but the named file does not exist, it is created with type $ and block size 1ØØH. Default output files are information-protected; user-specified output files created by DO will not have this attribute.

The output file, whether it is a named file or the default file, may be executed with the SETIN command. It is advisable to change the name of the file $DO.CMDØ if its contents are to be saved for repeated execution; if the name is not changed, the next execution of the DO program (apart from recursion or the any other execution of DO within a macro) will cause the contents of the file to be overwritten with a new output file. If multiple output files have been created, only the first need be executed with SETIN; the $DO.CMDØ file, because it contains the other DO commands, will recreate those files during its execution. (Because a DO output file is in standard text-format, it may be edited or printed like any other text file.)

S=options

These options may be specified as a string after the equals sign (=):

-X	prevents the output file from becoming the CI input file automatically after processing. If the user specifies the output file, this is the default condition.
X	causes the output file to become the CI input file automatically after processing. If the default output file is used, this is the default condition.
+X	Same as X
-A	indicates that the input file is not in ALS-8 format; this is the default condition.
A	indicates that the input file is in ALS-8 format.

EXAMPLES:

```
DO O=ASM,MACRO,5,BCDE
```

causes macro instructions to be written to a file called ASM. "MACRO" is the original input file, and both "5" and "BCDE" are actual parameters to be substituted for variables in that file. Because the output file argument is included in the command, ASM will not be executed automatically, that is, it will not be made the CI input file. If the command were:

```
DO O=ASM,S=+X,MACRO,5,BCDE
```

the ASM file would be made the console input file, because of the option specified. In either case, ASM can be executed later with the SETIN command.

4.4 EXAMPLES OF MACROS

The following examples illustrate many of the features of the macropreprocessor. In all examples, the symbol <cr> is a carriage return. Commands that begin with the dollar sign ($) are described near the end of Section 2 of this manual.

1) This is a macro input file called EX4$:

```
$REM [%1 IS PARAMETER 1]<cr>
[$REM %2 IS THE SECOND PARAMETER]<cr>
$STOP<cr>
```

The command DO EX4$,,FRED produces:

```
$REM<cr>
$REM FRED IS THE SECOND PARAMETER<cr>
$STOP<cr>
```

The command DO EX4$ produces:

```
$REM<cr>
$STOP<cr>
```

The second $REM is not present, because it is within the conditional.

2) This is a macro input file called MAKE$:

```
$NLST<cr>
$PR<cr>
$PR DOING %Ø  (Create and edit %1)<cr>
$REM DEFAULT TYPE IS "$"; DEFAULT BLOCK SIZE IS 1ØØH.<cr>
CREATE %1,[%2][%-2$],[%3][%-31ØØ]<cr>
$PR %1 HAS BEEN CREATED<cr>
$PR<cr>
$PR PRESS RETURN TO EDIT<cr>
$WAIT<cr>
EDIT %1<cr>
$PR JOB COMPLETE.<cr>
$LST<cr>
$STOP<cr>
```

The command line DO MAKE$,TEMP,,9CØ;FILES TEMP will result in the following output to the console:

```
DOING MAKE$  (Create and edit TEMP)
TEMP HAS BEEN CREATED

PRESS RETURN TO EDIT

(executes the PTDOS EDIT command and then returns to the macro)

JOB COMPLETE
$STOP
```

At this point control is returned to the Command Interpreter, which reads and initiates execution of the FILES command. The output from that command will show that a file called TEMP has been created with type "$" and block size 9CØH. (Notice that the only command that is echoed to the console during the execution of the macro is $STOP, because the first command in the macro turns off the PTDOS echo switch, and the command before $STOP turns it back on.)

3) This is a macro input file whose purpose is to change the attributes of as many as ten files at once. The file is called REAT, and is an example of the use of recursion.

```
$REM Stop if there are no more files<cr>
[%-1 $STOP]<cr>
[REATR %1,+I]; $REM Now REATR the rest of the files.<cr>
DO REAT,[%2],[%3],[%4],[%5],[%6],[%7],[%8],[%9],[%A]<cr>
$STOP<cr>
```

The command DO REAT, ONE, TWO, THREE, FOUR will cause the files ONE, TWO, THREE, and FOUR to be given the I (information protect) attribute.

SECTION 5

FILE SYSTEM

5.1 INTRODUCTION

PTDOS is, first and foremost, a system for managing files. Viewed abstractly, a file is a collection of data that for some reason belong together,

Most PTDOS files reside on a floppy disk. The detailed format of disk files will be discussed later in this section. In addition, PTDOS provides a mechanism, the "device file," that enables peripheral hardware devices to be manipulated by the system in the same manner as disk files. Further discussion of device files will be deferred until Section 9.

5.2 FILE CHARACTERISTICS

5.2.1 File Names

A PTDOS file resides on a single diskette and has a name that must be unique among all files on that diskette. A file name consists of 1-8 ASCII characters, excluding the following:

NAME	HEX ASCII CODE	GRAPHIC SYMBOL
control characters	00-1F	
space	20	
number sign	23	#
comma	2C	,
slash	2F	/
semicolon	3B	;
less than	3C	<
equals	3D	=
greater than	3E	>
DEL	7F	

Upper and lower case alphabetic characters are equivalent, and the high order (parity) bit in a character byte is ignored.

Because a diskette may be placed in any of the disk units in a system, the name of a file must usually be qualified by a "unit number". This number, denoted by an ASCII digit in the range 0-7, identifies the disk drive unit in which the diskette containing the file resides. A unit number, always preceded by a slash (/), is affixed to the end of a file name; for example, FILE/1 is a file that exists on a diskette in unit 1. If the name of a file does not have a unit number affixed to it, that file is presumed to reside on a diskette in the default unit.

5.2.2 File Types

A PTDOS file has a type as well as a name. It is common to assign the same file type to each of a group of files that have a common property, e.g., assembly language source files, or executable image files.

The file type is encoded in a single byte. File type 255 (FFH) is reserved to designate a device file (see Section 9). The remaining file types have no significance to the system, except that the high order bit is used to distinguish between executable image files and non-executable data files. If the high order bit is set, the Command Interpreter assumes that the file is not executable and will not attempt to load and execute it; if the high order bit is not set, the Command Interpreter assumes that the file can be loaded and executed (with unpredictable results if the file is not, in fact, an image file).

So far, the following file types have been assigned meanings:

FILE TYPE		DESCRIPTION
Hex	Symbolic	
ØØ	IØØ	System Image File
43	IC	System Command Program
47	IG	Game Program
53	IS	System Program (e.g., ASSM)
54	I$	System $Command File
2A	I.	Default Image File
8Ø	ØØ	System Data File
81	Ø1	Focal Binary Data File
82	Ø2	BCD Data File
83	Ø3	Focal Program
84	Ø4	BASIC/5 Program
85	Ø5	BASIC Program, Semi-Compiled Form
86	Ø6	BASIC Program, Text Form
87	Ø7	BASIC Serial Access Data File
88	Ø8	BASIC Random Access Data File
A4	$	Command File
AE	.	Default Data File
C1	A	SAVE Archive File

The symbolic form of the file type is the representation that appears in the listing produced by the FILES command. For example, a device file is type "D," and an image file is type "I" (or "I" followed by another character). The symbolic types consisting of a single printing character, or of "I" followed by a single character, may be used in the argument list of the CREATE, RETYPE, FILES, SAVE, and GET commands. Types with two digits in place of the single character can be included in an argument list only if represented by the control character whose ASCII code corresponds to the two digits.

5.2.3 File Protection Attributes

A set of protection attributes may be associated with a file to prevent certain operations from being carried out on that file. The protection attributes are encoded in a single byte whose bits, if set to 1, have the following meanings:

BIT	NAME	PROTECTION
7		Undefined, reserved for future use
6	PALO	Additional allocation of disk blocks
5	PNAT	Name and type change
4	PATR	Protection attribute change
3	PFINF	File information display
2	PREA	Read
1	PWRI	Write
Ø	PKIL	Kill

The names in the table are symbols defined in PTDEFS (see Section 6.3); the value of each symbol has the bit for a particular protection attribute set to 1, and all other bits set to Ø.

5.3 DISK STRUCTURE

Before it is possible to discuss file structure, it is necessary to describe the physical disk structure upon which the PTDOS file system is based. The surface of a diskette is divided into 77 circular tracks. In a Helios II system, each track is further divided into 16 sectors. Each sector can contain 256 bytes of data, and is separated from its neighbors by an intersector gap approximately 64 bytes in length.

The Helios II controller is capable of reading and writing physical blocks whose length ranges from 1 to 4095 bytes. A physical block always starts at the beginning of a sector, and its data are recorded contiguously; if a block is longer than 256 bytes, part of its data is written on one or more of the intersector gaps.

This recording technique, called "firm sectoring," has the advantage that more data can be written on the disk if block sizes are appropriately chosen. In order to maximize the space gained by writing data on the intersector gaps, a block must end exactly at the end of its last sector. This consideration dictates a choice of block sizes from the following list of optimum sizes:

HEX BLOCK SIZE	SECTORS PER BLOCK	SPACE GAINED (BYTES)
100	1	0
240	2	64
380	3	128
4C0	4	192
600	5	256
740	6	320
880	7	384
9C0	8	448
B00	9	512
C40	10	576
D80	11	640
EC0	12	704
FFF	13	768

The most efficient storage possible is achieved if a track is divided into exactly two blocks; the space gained on a track divided in this way is equivalent to 3 sectors. In practice, of course, it is not possible to format an entire diskette with 2-block sectors, so the actual space gained by firm sectoring always falls below the theoretical maximum.

A physical block is preceded by a header that has the following format:

#BYTES	DEFINITION
1	Sector number
1	Track number
2	Forward block chaining pointer, sector
	" " " " , track
2	Backward block chaining pointer, sector
	" " " " , track
2	File ID
1	Block length in sectors
2	Block length in bytes
2	Undefined, reserved for future use

The block chaining pointers and file ID will be discussed in the following section.

5.4 FILE STRUCTURE

A PTDOS file is a doubly linked list of physical blocks that reside on a single diskette. All blocks of a file have the same length. The list structure is implemented with the block chaining pointers in the block headers. The forward pointer in a block points to the next block of the file, while the backward pointer points to the preceding block. The first block of a file has a backward chaining pointer whose value is FFFFH, and the last block of a file has a forward chaining pointer whose value is the number of bytes actually used in that block plus 8000H.

This linked list structure makes it possible to access a file sequentially by following the pointers from block to block.

A file that is to be accessed randomly, as well as sequentially, must have an index block associated with it. This 256-byte block contains 2-byte entries pointing to the sector and track of each of the first 128 blocks of the file, or to every block of the file if it contains no more than 128 blocks; unused index entries are 0000H. Any byte in the first 128 blocks of an indexed file may be located by calculating the number of the block in which it must reside and then locating that block by means of the index. An index block header contains the ID of the file that it indexes, and has chaining pointers identical to the first block of that file.

Each file on a diskette has a unique file identification number (file ID). This number is assigned when the file is created and appears in the header of every block of the file. Every diskette contains a file named NEXTID, or contains the 2-byte value of the next unique file ID that can be assigned; the value is updated every time a file is created. The file ID is 16 bits long, so no more than 65,536 files may be created during the life of a diskette; in the unlikely event that this limit is reached, the files on that diskette should be copied to another diskette on which it is still possible to create files. The original diskette can then be reused if it is conditioned or formatted with DISKCOPY.

Every PTDOS diskette has a directory containing information about the files residing on that diskette. The directory is itself a file named DIRECTRY. It occupies all of track 25 (19H), and consists of 16 256-byte blocks, each of which has the following format:

#BYTES	DEFINITION
1	Number of entries in block (0-12)
1	Pointer to next free byte in block
.	
.	Up to twelve 21-byte entries
.	
2	Unused

Each directory entry has the following format:

#BYTES	DEFINITION
8	File name, filled on right with NULs (00H)
1	File type
2	Block size in bytes
1	File protection attributes
2	File ID
2	Pointer to index block, sector " " " " , track
2	Pointer to first block of file, sector " " " " " " , track
2	Number of blocks in the file
1	Undefined, reserved for future use

The index block pointer has the value Ø if the file has no index block.

Because each block can contain at most 12 entries and there are 16 blocks in the directory file, no more than 192 files can exist simultaneously on a diskette.

5.5 DISK SPACE ALLOCATION AND THE FREE SPACE MAP

Initially, all free space on a diskette is formatted into free blocks exactly one sector (256 bytes) long with a file ID of Ø and chaining pointers appropriate for both the first and last block of a file (i.e., a backward pointer of FFFFH and forward pointer of 8ØØØH). The location of all free sectors is represented in a free space bit map, a file named FSMAP that contains 77 2-byte entries, one for each track. The 16 bits of each track entry correspond to the 16 sectors of that track; if a bit is 1, the corresponding sector is free.

When a file is created or data is added for which there is insufficient space in the last block of the file, another block must be allocated. This allocation is accomplished by searching the free space map for the first area on the disk where there is a sufficient number of contiguous free sectors to hold a block of the requisite length. The bits in the free space map that correspond to these sectors are then set to zero, and the empty block is created and linked to the block that was formerly the last block of the file.

When an end of file is placed in a block that was not previously the last block of the file, or when an entire file is killed, one or more blocks of the file can be returned to the free space pool. This recovery is accomplished by reformatting the disk area occupied by the block into one or more free 256-byte sectors and setting the corresponding bit(s) in the free space map to 1.

5.6 FILE ACCESS AND BUFFERING

To access a file, a program must make a series of system calls to open the file, read data from and/or write data to the file, and finally close it. There are also system calls to space over part of a file, endfile a file, and seek to a particular location in a random access file before reading or writing. These file access calls will be described in detail in Section 7, in addition to the system calls that create or kill a file, change its name, type, and attributes, and create an index block for a file that is to be accessed randomly.

When a file is opened, a number of actions are taken by the system:

1) A file number is assigned. This number will be associated with the file only as long as it is open; once that file is closed, the same file number may be assigned to another file when it in turn is opened.

2) A file control block (FCB) is allocated. This contains, along with other information about the file, a pointer to the byte at which the next read, write, endfile, or space operation will begin. This

pointer is called the "file cursor", and initially points to the first byte of the file; it is updated by the read, write, space, and seek operations.

3) A buffer is allocated whose length is equal to the block size of the file. The first block of the file is read to "prime" the buffer.

All transfers to and from a file are buffered by PTDOS. When a program executes a read operation, the data are transferred from the file buffer into memory designated by the program. If some data required are not in the buffer, then another block of the file is read into the buffer. The transfer continues in this manner until complete. When a program is executing a write operation, the data are transferred into the buffer, which is written to the file only when necessary. Any transfer into the buffer sets a "dirty flag" indicating that it contains data not yet present in the file. A dirty buffer will be written to the file anytime that another block is to be read into the buffer. (Writing out a full buffer when extending a file may be considered a special case in which the new block to be read is empty.)

PTDOS allows a choice of buffer management technique. Normally, buffers reside in system-managed memory and may have static or dynamic allocation. Buffers may also reside in memory managed by the user program, in which case their allocation is always static.

A static buffer is allocated when a file is opened, and continues to exist until that file is closed. In contrast, a dynamic buffer is allocated each time a request is made to execute a read or write operation, and is deallocated immediately thereafter. The advantage of dynamic buffering is that when the memory available for file buffers is limited, several open files may all use the same memory area for their buffers; the disadvantage is that the system's I/O overhead is increased and most file operations will require many more time-consuming data transfers between the buffer and the device on which the file resides (usually a diskette).

5.7 DATA STRUCTURE

5.7.1 Introduction

Thus far, the physical structure common to all files has been discussed. This physical structure is the only structure of which the PTDOS file system is aware; any structure imposed on the data contained in a file is meaningful only to the program or programs that are intended to manipulate that file.

This section will describe the data structure of files manipulated by the layers of PTDOS that are built on top of its file system.

5.7.2 Text Files

The text files used, for example, by EDIT, EDT3, and PRINT have a very simple structure. Such a file is just a string of characters with an ASCII carriage return character (CR or ØDH) marking the end of each line of text. A blank line is represented by a carriage return

immediately following the carriage return that terminated the previous line. In addition to this line structure, a file produced by EDT3 may be divided into pages; each page is terminated by an ASCII form feed character (FF or 0CH).

EDIT can also read and write the ALS-8 text file format, and convert files from ALS-8 to standard format, and vice-versa. Lines in an ALS-8 file, in addition to being terminated by a CR, are prefixed by a 1-byte binary length field. A blank line is thus 2 bytes long, counting its terminating CR.

5.7.3 Image Files

An image file contains the machine language code for a program, the address at which the Command Interpreter is to load it, and usually the address at which execution of the loaded program is to begin. An image file consists of zero or more segments that have the format:

#BYTES	DESCRIPTION
2	Number of code and data bytes in segment
2	Load address of code and data belonging to the segment
Variable	Code and/or data

The CI will load each segment at the specified address until the end of file is reached or a starting address is encountered.

A starting address usually is represented as two bytes immediately following the last segment of the file, i.e., as the last two bytes of the file. It is also possible to represent a starting address as a segment with a zero byte count:

#BYTES	DESCRIPTION
2	Identically zero
2	Starting address

The two representations for a starting address differ only in that if the first representation is encountered, the image file is closed after it has been loaded, whereas if the second form (zero length segment) is encountered, the image file is left open after it has been loaded.

ASSM, the assembler, produces image files with a maximum segment byte count of 100 bytes and uses the first representation for a starting address if an XEQ pseudo-operation appears in the assembly language source program.

5.7.4 Utility Files

The files created by BLDUTIL and accessed by the utility handler consist of a directory table followed by one or more program or data

modules, each of which has the format of an image file. The directory table contains offsets from the first byte of the utility file to the beginning of each module, so that the utility handler can locate and load any module. If a module has a starting address, that address must be represented as a segment with a length of zero, i.e., in the format described in Section 5.7.3.

A utility file has the following format:

#BYTES	DESCRIPTION
1	Number of modules in the file
2	Offset to module Ø
:	
:	
2	Offset to last module
8	Name of module Ø
Variable	Module Ø, image format
:	
:	
8	Name of last module
Variable	Last module, image format

Notice that the offsets are to the first byte of the code and data of each module, not to its name.

SECTION 6

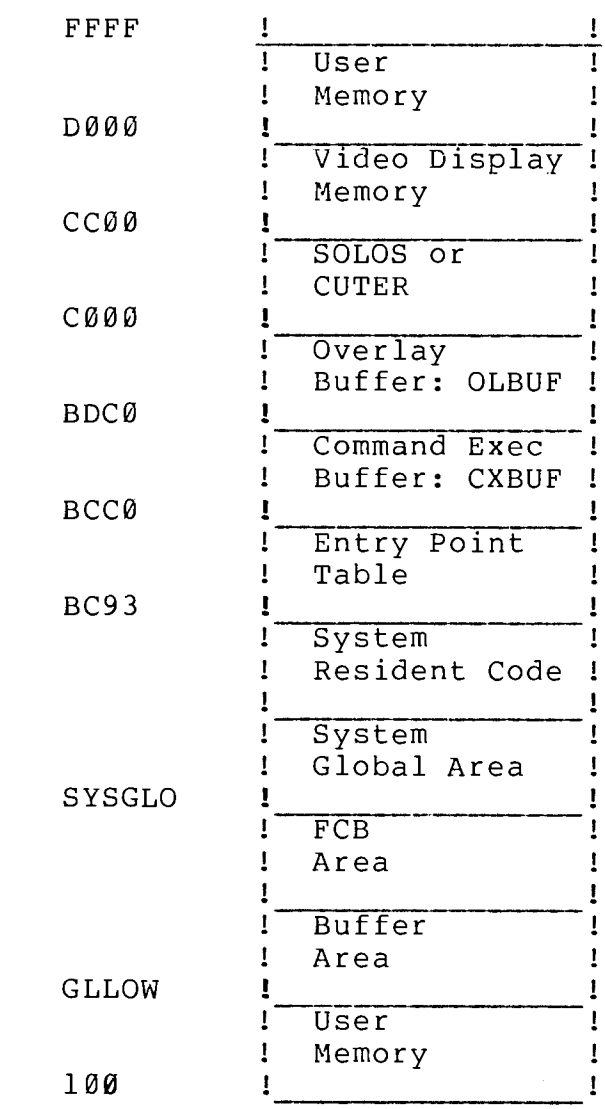
SYSTEM INTERFACE

6.1 INTRODUCTION

This section explains the interface between PTDOS and a user-written program. Details of the system calls, system utilities, and device drivers appear in Sections 7-1Ø; this section provides necessary background for those discussions.

6.2 MEMORY MANAGEMENT

PTDOS extends downward in memory from location BFFFH to the address specified in the system global parameter GLLOW (see Section 6.3). In a Sol the area from CØØØH to CCØØH is left free for the SOLOS moritor program; in a machine other than a Sol, this area may be used by the CUTER monitor program. The area from CCØØH to DØØØH is normally used as the video display memory of the Sol or VDM-1. The areas from Ø to GLLOW and from DØØØH through FFFFH are available as user memory. The system memory map may be represented as follows:



When a file is opened, the system must allocate a file control block (FCB) and a buffer for it (see Section 5). File control blocks are allocated in the FCB area; this area has a fixed size that depends on the maximum number of open files allowed by the system configuration. Buffers are allocated in the memory between the FCB area and GLLOW. The amount of memory available for buffers is determined by the value of GLLOW, which may be set by the user with the SET BU= or CONFIGR command. The CONFIGR command sets GLLOW in the disk-based resident and thus determines the default value that will be in effect whenever the system is bootstrapped. If sufficient memory is available, 8800H is a good choice for the default value, because it allows two files with 4C0H block sizes be open simultaneously. A value of 8000H will serve all but the most extreme needs, e.g., unusually large block sizes or number of open files.

It is possible for the user to protect the lower part of memory by setting the system global parameter GLPRO. If this parameter has a non-zero value, PTDOS will not allow memory from 0 to GLPRO to be used as a user buffer or as a data destination for a read or delimited read operation.

When possible, system commands are executed in the command execution buffer (CXBUF); such commands are called "safe." The Explain Error Utility (see Section 8.2) runs in the overlay buffer (OLBUF); hence, it can be invoked by the safe commands that run in CXBUF. Commands whose code will not fit in CXBUF are executed in user memory starting at 100H; the memory below 100H is never used by the system.

6.3 SYSTEM GLOBAL AREA

The System Global Area is a user-accessible area of memory containing various parameters that control the operation of PTDOS. These parameters appear in the following order and have the specified functions. A number in parentheses is a cross-reference to additional information about a parameter.

Symbol	#Bytes	Description
GLCIF	1	CI input file number (3.3.1)
GLCOF	1	CI output file number (3.3.1)
GLUTF	1	Utility file number (6.4.5)
GLECF	1	CI Echo file number (3.3.1)
GLUNI	1	Default unit number
GLPRM	1	Number of permanently open files
GLMXU	1	Maximum number of units
GLSWI	2	User sense switches
GLERS	2	Level 2 error trap address (6.5)
GLERM	2	Level 1 error trap address (6.5)
GLERH	2	Level 0 error trap address (6.5)
GLTRP	2	CI return trap address (3.3.2)
GLPRO	2	User memory protect address (6.2)
GLFLG	1	Console character waiting flag (9.4.3)
GLBYT	1	Console waiting character (9.4.3)
GLRCH	2	Console read character routine pointer (9.4.3)
GLWCH	2	Console write character routine pointer (9.4.3)
GLTCH	2	Console test for waiting character routine pointer (9.4.3)
GLNCT	1	Console number of nulls to follow LF (9.4.3)
GLVER	1	System version number
GLDAT	3	System date Format is packed BCD, MMDDYY.
GLNAM	8	Disk name Format is ASCII, zero filled on right.
GLPAS	8	Password used by CONFIGR command (2.2) Format is ASCII, zero filled on right.
GLBD1	2	Pointer to wait for interrupt routine (6.6)
GLBD2	2	Reserved for future use
GLIF1	1	Interrupt enable flag (6.6)
GLIF2	1	Disk interrupt processing complete (6.6)
GLLOW	2	Lowest address assigned to system (6.2)
GLECH	1	CI echo enable flag (3.3.1)
GLLOK	1	Disk write lock flag
GLUPS	1	Console ASCII upshift flag (9.4.1)
GLRBC	1	Disk read-after-write-check flag
GLBIO	1	Console binary I/O flag (9.4.1)
GLLOG	1	CI log file enable flag (3.3.1)
GLVRB	1	Verbose flag used by BOOTLOAD command (2.2)

The symbolic names given in the table are offsets to the parameters from the beginning of the System Global Area. These names are defined by EQU assembler pseudo-operations in either of two files appearing on the PTDOS System Diskette:

PTDEFS	assembly language definitions with line numbers;
NPTDEFS	assembly language definitions without line numbers.

These files are meant to be included in the source code of a user program by means of the COPY assembler pseudo-operation.

The starting address of the System Global Area is given by the parameter SYSGLO in the System Entry Point Table (see Section 6.4). To access a system global parameter, a program must add this starting address to the offset for the parameter. For example,

```
LHLD  SYSGLO   get starting address of System Global Area
LXI   D,GLCIF  add offset to CI input file number
DAD   D
MOV   A,M      get value of CI input file number
```

6.4 SYSTEM ENTRY POINTS

All system entry points available to a user program are organized into a System Entry Point Table. The starting address of this table (BC93H) will not be changed in future releases of PTDOS. Because the starting address of the System Global Area is subject to change in future releases, that address is contained in SYSGLO (at location BCA5H) in the Entry Point Table (see Section 6.3).

The System Entry Point Table is summarized below. The entry points will be discussed in detail in the sections that follow.

PTDEFS Symbol	Hex Address	Description
SYS	BCBC	Standard PTDOS operation entry point
RB	BCB9	Read single byte from file
WB	BCB6	Write single byte to file
UTIL	BCB3	Utility Handler
SRESET	BCBØ	Short reset entry point
ERRLØ	BCAD	Initiate error level Ø return to a user program
ERRL1	BCAA	Initiate error level 1 return to a user program
ERRL2	BCA7	Initiate error level 2 return to a user program
SYSGLO	BCA5	Pointer to the System Global Area
CONIN	BCA2	Read single character from console
CONOUT	BC9F	Write single character to console
CONTST	BC9C	Test console for waiting character
PSCAN	BC99	Parameter Scanner
INTDN	BC96	Disk interrupt processing complete entry point
INTDK	BC93	Disk interrupt processing entry point

The symbolic entry point names given in the table are defined in the files PTDEFS and NPTDEFS. The value of each symbol is the address of the corresponding entry point.

6.4.1 SYS

Most system calls are made through the SYS entry point. The calling sequence for SYS is as follows:

```
<load all necessary registers>
CALL SYS
DB <operation code>
JMP <error routine>
<normal return point>
```

The possible operations are:

Code Dec	Code Hex	PTDEFS Name	Function
Ø	Ø	CREOP	Create file
1	1	OPEOP	Open file
2	2	KILOP	Kill file
3	3	RBLOP	Read data
4	4	WBLOP	Write data
5	5	SPAOP	Space file cursor
6	6	EOFOP	Endfile file
7	7	CLOOP	Close file
8	8	CHTOP	Change file type
9	9	CHAOP	Change protection attributes
1Ø	A	CHNOP	Change file name
11	B	INFOP	Request file information
12	C	SUNOP	Set default unit
13	D	RETOP	Return to system
14	E	RESOP	Reset and return
15	F	ABTOP	Abort and return
16	1Ø	SEKOP	Seek to specified location
17	11	RNDOP	Randomize file
18	12	CAOP	Close all files
19	13	CTLOP	Control/Status
2Ø	14	RESOP	Short reset
21	15	DRDOP	Delimited read
22	16	DWROP	Delimited write
23	17	DSPOP	Delimited space
24	18	RTROP	Return to system with trap set

The operations will be discussed individually in Section 7. The registers to be loaded prior to the call depend upon the operation requested. SYS requires four bytes of the user program stack, in addition to the two bytes used by the CALL instruction.

If no errors occur, all operations except RETOP, RESOP, ABTOP, and RTROP return to the normal return point in the calling program. The values of the registers depend upon the operation that has been completed.

If an error is detected while a system call is being serviced, and if the trap address for that error level is set to -1, a return is made to the error return point, the location immediately following the

operation code (see Section 6.5). The error return point normally contains a JMP to an error-handling routine in the user program. On an error return, register A contains the error code; the values of all other registers depend upon the operation that was being performed.

The error returns possible from each operation are discussed in Section 7. If a SYS call is made with an illegal operation code, a level 1 error return is made with error code ØAH (ERIOP).

6.4.2 RB, WB

The RB and WB entry points allow a single byte to be read from or written to a file with minimal overhead. These entry points are somewhat more efficient than the read and write SYS calls for transfers of a single byte. However, this decrease in overhead is achieved only if very few bytes are transferred to or from a block; otherwise, it is more efficient to use the normal read and write SYS calls. The potential gain in efficiency is entirely negated if the file being accessed with RB or WB has been opened with dynamic buffering.

Both RB and WB run on the calling program stack and require 3Ø bytes of stack space in addition to the two bytes used by the CALL instruction.

The calling sequence for RB is as follows:

```
        <load register A with file number>
        CALL RB
        JMP <error routine>       error code is in register A
        <normal return point>     byte read is in register A
```

If no errors occur, a return is made to the normal return point. The character read will be in register A; the values of all other register will be undefined.

If an error occurs, and if the trap address for that error is set to -1, a return will be made to the error return point, the location immediately following the operation code (see Section 6.5). The error return point normally contains a JMP to an error-handling routine in the user program. On an error return, register A contains the error code; the values of all other registers are undefined. Possible level 1 errors are:

Code		PTDEFS	Description
Dec	Hex	Name	
3	3	ERUFN	File not open
4	4	ERPRO	Protected file
24	18	EREOF	End of file
27	1B	ERMOV	No space for buffer

The calling sequence for WB is as follows:

```
<load register A with file number>
<load register B with byte to be written>
CALL WB
JMP <error routine>      error code is in register A
<normal return point>
```

If an error occurs and a return is made to the error return point, register A contains the error code; the values of all other registers are undefined. Possible level 1 errors are:

Code Dec	Code Hex	PTDEFS Name	Description
3	3	ERUFN	File not open
4	4	ERPRO	Protected file
27	1B	ERMOV	No space for buffer
29	1D	ERDFL	Disk full

6.4.3 SRESET, ERRLØ, ERRL1, ERRL2

Each of these four entry points may be called by a user program in response to certain error conditions. If such a call is made, control will not return to the calling program.

SRESET behaves exactly like the RESOP call to SYS (see Section 7), that is, it causes a "reset" invocation of the Command Interpreter (see Section 3.4). Its calling sequence is simply:

```
CALL SRESET
```

The ERRLØ, ERRL1, and ERRL2 entry points are called by a device driver to initiate a level Ø, 1, or 2 error return to a user program (see Section 9.2.1). The calling sequence for all three is exemplified by:

```
CALL ERRLØ
DB <error code>
```

The specified error code will be returned to the user program in register A.

6.4.4 CONIN, CONOUT, CONTST

These entry points are used for single character console I/O. Their operation is controlled by several system global parameters; Sections 9.4.1 and 9.4.2 contain a full description.

CONIN reads one character from the console input device. Its calling sequence is:

```
CALL CONIN
<process the character in register A>
```

On return, the character read is returned in register A. CONIN will not return until it has read a character; if none is available, it waits until one is.

CONTST checks whether there is a character waiting to be read. A program that does not wish to wait for a character if none is available should call CONTST prior to calling CONIN. The calling sequence is:

```
CALL CONTST              check for waiting character
JZ <somewhere>           if none, do something else
CALL CONIN               input the character
<process the character>
```

On return from CONTST, the zero flag will be set if no character is available; otherwise, the flag will be cleared.

CONOUT writes one character to the console output device. Its calling sequence is:

```
<load register A with the character to be output>
CALL CONOUT
```

CONOUT returns after the character has been output.

6.4.5 UTIL

UTIL is the entry point to the Utility Handler, which may be called by a user program to load overlays. Overlays are pieces of a program (code and/or data) that share memory space; a program divided into overlays can be larger than the real memory space available to run it.

In PTDOS, overlays take the form of modules in a "utility file." A utility file is a random access file (i.e., it has an index block, see Section 5.4) that consists of a collection of modules preceded by a directory table. Utility file structure is discussed in detail in Section 5.7.4. BLDUTIL, the system command used to build and modify utility files, is explained in Section 2.2.

The modules in a utility file are identified by module numbers; the first module is number 0, and the largest possible module number is 255. This number is used to specify the module to be loaded by a call to the Utility Handler. The utility file from which the module is to be loaded must be open, and its file number must be placed in the system global parameter GLUTF that identifies the current utility file.

UTIL has the following calling sequence:

```
<load registers A, H, and L>
CALL UTIL
DB <module number>
JMP <error routine>
<normal return point>
```

The Utility Handler requires four bytes of the calling program stack, in addition to the two bytes used by the CALL instruction. Information may be passed to the loaded module in registers A, H, and L.

Because the normal return point from a UTIL call is three bytes after the return address stored on the stack by the CALL instruction, the loaded module must increment that address three times prior to making the return. The following code will accomplish this:

```
XTHL      swap HL and the return address
INX H     triply increment the return address
INX H
INX H
XTHL      swap back
RET       return to the normal return point
```

If an error is detected, the Utility Handler returns to the error return point, the location immediately following the module number. The module can make an error return to the calling program by executing a RET instruction (or equivalent).

On either a normal or error return, the values of the registers are determined by the module that was called.

6.4.6 PSCAN

PSCAN is the entry point to the Parameter Scanner, a routine intended to support the processing of arguments by any program that runs under PTDOS. PSCAN accepts input from either the CI input file or a buffer in user memory, and can process that input in three ways:

1) PSCAN can copy one or more characters from its input source into a user output buffer for further processing by the user program.

2) If presented with a file name, PSCAN can create and/or open the file.

3) If presented with the ASCII representation of a numeric value in any base, PSCAN can convert it into its internal binary representation. As a special case, PSCAN will convert an ASCII file number (e.g., #2) into its internal binary equivalent.

PSCAN runs on the user stack and has the following calling sequence:

```
<load A with operation code>
<load B with base, if required>
<load DE with address of output buffer>
<load HL with address of input buffer, if required>
CALL PSCAN
```

The operation codes recognized by PSCAN are:

Hex Code	PTDEFS Symbol	Description
4	PSC	Create file if it does not exist
Ø	PSCO	Create file if it does not exist, open file
1	PSOP	Open file
6	PSFC	Create file (must not exist already)
2	PSFCO	Create and open file (must not exist already)
A5	PSONE	Read single character
8	PSOPT	Read characters until delimiter is encountered
5	PSN	Read file name
85	PSV	Convert numeric value
D5	PSCN	Convert numeric value in user buffer using base in B

The PSCN operation (D5H) gets its input from a buffer in user memory. The rest of the operations all accept input from the CI input file. However, any of them will accept input from a user buffer if 4ØH is added to the operation code in the table. For example, operation code 41H (1 + 4ØH) instructs PSCAN to open a file whose name is in user memory. Whenever a user input buffer is to be used, register pair HL must be set to point to the first byte of the buffer before PSCAN is called.

For all operations, register pair DE must point to a 2Ø-byte output buffer in user memory. Input characters are transferred into this

buffer as PSCAN reads them. For any operation that may create a file, the output buffer must be preceded by a file information block with the format:

#Bytes	Description
1	File type
2	Block size
1	Protection attributes

File names input to PSCAN must adhere to the requirements stated in Section 5.2.1. They may be terminated by any of the delimiter characters: comma, semicolon, carriage return, or NUL (ØØH).

On a normal return from processing a file name, the output buffer will contain the name followed by a NUL (ØØH). If a file was opened, its file number will be in register E. If an illegal character appears in a file name, or if 2Ø characters appear without a valid delimiter, the scan will be terminated immediately. In the first case, PSCAN will return to the calling program with the Carry flag cleared and a -1 in register E. In the second case, it will return with the Carry flag set and a Ø in register E. If the scan is terminated by either a valid delimiter or an illegal character, on return register C contains the number of characters scanned.

If a file number rather than a file name appears in the input to PSCAN, no action is taken except that the decimal ASCII file number is converted into internal binary in register E. A file number has the format #n, where n is a decimal number. If an illegal character appears in a file number, PSCAN will return to the calling program with the Carry flag set and a Ø in register E.

A numeric value input to PSCAN must have the format

```
number{:base}
```

and must be terminated by a NUL (ØØH). The base qualifiers recognized by PSCAN are:

B	Binary
O or Q	Octal
D	Decimal
H or none	Hexadecimal

If the operation is PSCN (D5H), the numeric value may not include a base specifier; instead, the binary value of the base (e.g., 2, 8, 1Ø, 16) must be placed in register B before PSCAN is called.

The number part of a numeric value is unsigned and is constructed from that subset of the digits Ø-9,A-F appropriate to its base. It may be no larger than the largest 16-bit quantity, i.e.,

```
1111 1111 1111 1111:B
177 777:Q
65535:D
FFFF:H
```

On a normal return from a value conversion operation, the binary representation of the value will be in register pair DE. The output buffer will contain the ASCII input value terminated by a NUL (ØØH); if a base qualifier is present, the colon will be replaced by a NUL in the output buffer. Register C will contain the number of characters scanned. If 2Ø characters appear in the input without a valid delimiter, the scan will be terminated immediately. In that case, or if the value is too large or contains an illegal base qualifier or a character not legal for its base, PSCAN will return to the calling program with the Carry flag set and a Ø in register E.

On any return from PSCAN, register A contains the last character scanned; on a normal return, the last character will be one of the valid delimiters, except in the case of the Read Single Character operation. If the first character read by PSCAN is a delimiter, it will return with the Zero flag set and the Carry flag cleared. If any PTDOS error occurs, PSCAN will return to the calling program with the Carry flag set and the error code in register E.

6.5 ERROR HANDLING

Errors detected by PTDOS or by a program running under PTDOS are characterized by a severity level:

Level Ø — These are very serious errors in disk or file structure, usually generated by a hardware failure.

Level 1 — These are less serious errors generated by programs that improperly request system services or request services for which there are insufficient resources (e.g., file does not exist, no more buffer space).

Level 2 — These are warnings that are generated by singular conditions arising during normal functioning of the system (e.g., end of file).

Errors at each level can be handled in one of three ways, as determined by the error trap addresses in the System Global Area:

GLERH	level Ø error trap address;
GLERM	level 1 error trap address;
GLERS	level 2 error trap address.

If any of these parameters has the value -1 (FFFFH), errors of the corresponding level cause a return to the error return point in the calling program. The error code will be in register A; all other registers will have the values they had prior to the system call except as noted in the discussions of the individual operations.

If any of the error trap addresses has the value Ø, errors of the corresponding level cause an ABORT system call with the error code in register A, the operation code in register B, and the address of the system call in register pair HL (see Section 7). This call results in a console message of the form:

ERR: <error code> <operation code> <address of call>

The abort handler then waits for a key to be pressed. If the key is not a carriage return, a call is made to the Explain Error Utility, which displays the error message that corresponds to the error code. If the level Ø abort is generated by the disk driver, the following message is output to the CI output file before control is passed to the system error handler:

DSK: <file ID> <sector,track> <header ID> <header block size>

If any of the error trap addresses has a value other than -1 or Ø, errors of the corresponding level cause control to pass to a trap routine at that address. Such a routine may take any action deemed appropriate. The error code will be in register A; all other registers will have the values they had prior to the system call except as noted in the discussions of the individual operations. When an error trap routine is entered, the stack will be in the state that existed immediately prior to the system call.

6.6 INTERRUPT PROGRAMMING

PTDOS can be run in a system with interrupt-driven devices. There are periods during disk transfers when instruction timing is critical. At the beginning of such periods, the disk driver executes an 8Ø8Ø disable interrupts instruction (DI). When the critical period has passed, the driver will re-enable interrupts with an 8Ø8Ø enable interrupts instruction (EI) if the following system global parameter has a non-zero value:

GLIF1	interrupt enable flag.

Because the area of memory from Ø to 1ØØH is not used by PTDOS, a user-supplied interrupt handler may reside there, allowing vectored interrupts to one of the 8Ø8Ø restart locations (Ø, 8, 1ØH, 18H, 2ØH, 28H, 3ØH, 38H).

It is possible to modify the Helios II controller so that interrupt signals are generated by the conditions "seek complete" and "transfer complete." Thus it possible to implement a scheduler that will run one or more tasks that do not use PTDOS during the disk operations requested by a task that is running under PTDOS. (PTDOS is not reentrant, so only one task can use it at a time.) In order to do this, GLIF1 must be set to a value other than zero. Two other system global parameters are significant:

GLIF2	disk interrupt processing complete flag;
GLBD1	pointer to wait for interrupt routine.

In addition, two system entry points are used by the scheduler and the interrupt handler:

INTDK	disk interrupt processing entry point;
INTDN	disk interrupt processing complete entry point.

When the PTDOS task has requested a disk operation and the system is waiting for completion of a seek or data transfer, it passes control to the scheduler routine whose address (non-zero) is in GLBD1. This scheduler should switch to its own stack, and can then start the other task(s) running.

Occasionally, the non-PTDOS task(s) should return control to the scheduler so that GLIF2 may be examined. If that parameter has a value of zero, one of them may continue. Otherwise, the scheduler should restart the PTDOS task by calling the INTDN entry point. PTDOS will switch back to its own stack before proceeding.

When an interrupt occurs, the interrupt handler must call INTDK. If the interrupt was generated by the disk controller, PTDOS services it and returns to the interrupt handler with the Carry flag set. If the interrupt was generated by some other device in the system, PTDOS returns with the Carry flag cleared; in this case, the interrupt handler is responsible for servicing the interrupt. In either case, the values of all registers are undefined on return from INTDK, and the interrupt handler is responsible for re-enabling interrupts with an 8080 enable interrupts instruction (EI).

SECTION 7

SYSTEM CALLS

7.1 INTRODUCTION

The purpose of this section is to describe each of the 25 system calls available through the SYS entry point to PTDOS. Each description consists of:

1) the name of the operation, its symbolic name (as defined by PTDEFS and NPTDEFS), its decimal code, and its hexadecimal code;
2) a brief statement of the purpose of the operation;
3) instructions for loading expected values into the registers before the call;
4) a more detailed discussion of the procedure involved in the operation;
5) a list of possible error returns, with their hexadecimal and decimal codes and the system error messages to which they correspond; and
6) additional notes, where appropriate.

This discussion assumes familiarity with the background material supplied in Sections 5, 6, and 9.

7.2 STANDARD NAME RESOLUTION RULES

Many operations require that a file name be specified. Names are represented as strings of 1Ø or fewer characters, including an optional unit number; such a string is scanned from left to right (lower to higher locations in memory) until a NUL (ØØH) or an ASCII slash (/) is encountered. A name is expected to consist of no fewer than 1 and no more than 8 characters. Several ASCII characters are not legal in file names; a list of these characters appears in Section 5.2.1. No distinction is made between upper and lower case letters, (i.e., "A" and "a" are equivalent), and the high-order (parity) bit in each character is ignored. If an illegal character is encountered in the string, or if the string consists of too few or too many characters -- that is, if a terminator does not occur after the eighth character -- an error is generated.

If the scan is terminated by a NUL, the named file is associated with the default unit specified by GLUNI in the System Global Area (see Section 6.3). If the scan is terminated by a slash (/), the next character must be an ASCII digit specifying the unit with which the file is to be associated. This digit must be less than the value stored in GLMXU in the System Global Area. (GLMXU contains the maximum number of units to be supported by the system; because unit numbers begin with Ø, the digit corresponding to the highest available unit has a value of GLMXU-1.) An error will be reported if the character supplied is not a unit in the acceptable range.

7.3 ERROR RETURNS

Possible error returns are discussed in the context of each operation. If no error occurs during an operation, the return to the calling program is "normal": that is, control passes to the address three bytes after the system call, and the CPU registers and flags, except Carry, are left intact (exceptions are noted). If an error does occur, an error return is made to the calling program: control is passed to the address immediately following the operation code, and a hexadecimal code corresponding to the error is returned in the A register (see Section 6.4.1).

Each of the errors detected by PTDOS or by a program running under PTDOS is assigned with a severity level (see Section 6.5). Level Ø (very serious) errors may occur during almost any operation and are therefore not discussed in relation to particular operations. These errors are associated with failure of the computer, disk surface, disk drive and controller, or system integrity. Examples of Level Ø errors are:

Message:	PTDEFS name:	Error codes: HEX	DEC
FILE ID CONFLICT	ERFIC	21	33
BLOCK SIZE CONFLICT	ERBSC	22	34
SECTOR CONFLICT	ERSCC	23	35
CAN'T FIND SECTOR	ERCFS	24	36

When a serious error occurs, a message of the following form is displayed on the CI output file:

DSK: <file ID> <sector,track> <header ID> <header block size>

before control is passed to the system error handler.

A procedure for dealing with hardware-related errors is outlined in Section 1Ø.

7.4 OPERATION DESCRIPTIONS

ABORT ABTOP 15 ØFH

PURPOSE: Return to the Command Interpreter and print an error message on the console. Terminate processing of the command string, and reset console input and output files to #Ø and #1, respectively.

REGISTERS ON ENTRY:

A contains the error number.
B contains any useful number that can be represented in 8 bits.
HL contains any useful number that can be represented in 16 bits.

PROCEDURE:

The message

ERR: AA BB HHLL

is printed on the console output device. In the example, "AA" represents the content of register A, "BB" represents the content of register B, and "HHLL" represents the content of the HL register pair at the time of the call.

After the message is printed, the user may enter a single character on the console input device. If that character is a carriage return, a RESET operation is performed (see the RESET system call later in this section). If the character is not a carriage return, a call is made to the system Explain Error Utility (UXOP).

If all goes well, i.e., if the utility file can be read, an English explanation of the error will be printed on the console, and a RESET will be performed. If an error occurs during the call to the Explain Error Utility, the message:

CAN'T EXPLAIN

will be printed, and another ABORT will be attempted.

NOTES: If the error number in A is zero, the explanation phase will be omitted and a RESET performed. If the error number does not correspond to one of the legal error codes, the English message will be

ERROR: ILLEGAL

There are no returns from this call.

CHANGE ATTRIBUTES CHAOP 9 9H

PURPOSE: Change the protection attributes of a file.

REGISTERS ON ENTRY:

DE points to the file name
H contains the desired new attributes

PROCEDURE: The directory is searched on the unit specified by the file name. If the file exists and is not protected against a change in its attributes, the new attributes are substituted for the old ones. (The attribute byte has a particular bit set for each desired attribute; see Section 5.2.3.)

Level 1 errors:	PTDEFS name:	Error codes: HEX	DEC
FILE DOES NOT EXIST	ERNEX	1	1
FILE IS PROTECTED	ERPRO	4	4

No level 2 errors

CHANGE NAME CHNOP 10 0AH

PURPOSE: Change the name of a file

REGISTERS ON ENTRY:

DE points to the current file name
HL points to the proposed file name

PROCEDURE: The directory is searched on the unit specified by the file name to ensure that the new name is not currently in use. The standard name resolution rules apply, and both names must resolve to the same unit. The file to be renamed must exist and must not be protected against a change in its name.

The new name is substituted for the old one, and the name change is recorded in the directory.

Level 1 errors:	PTDEFS name:	Error codes: HEX	DEC
FILE DOES NOT EXIST	ERNEX	1	1
FILE ALREADY EXISTS	ERAEX	2	2
FILE PROTECTED	ERPRO	4	4
UNIT CONFLICT	ERUCN	13	19

No level 2 errors

CHANGE TYPE CHTOP 8 8H

PURPOSE: Change the type of a file.

REGISTERS ON ENTRY:

DE points to the file name
H contains the new type

PROCEDURE: The directory is searched on the unit specified in the file name. If the file exists and is not protected against a change in its type (see Section 5.2.3), the new type is substituted for the old one, and the change is recorded in the directory on the diskette.

Level 1 errors:	PTDEFS name:	Error codes: HEX	DEC
FILE DOES NOT EXIST	ERNEX	1	1
FILE PROTECTED	ERAEX	4	4

No level 2 errors

CLOSE CLOOP 7 7H

PURPOSE: Terminate processing of a file and release resources associated with access to the file.

REGISTERS ON ENTRY:

A contains the file number assigned to the file when it was opened.

PROCEDURE: If a change has been made in the block currently in the buffer, the buffer is flushed to the disk. The space occupied by the buffer is released, and the file number is freed for reuse.

Level 1 errors:	PTDEFS name:	Error codes: HEX	DEC
ILLEGAL FILE NUMBER	ERIFI	32	50

No level 2 errors

NOTES:

Remember that some files cannot be closed. (The number of such files is determined by the GLPRM parameter in the System Global Area.) An attempt to close one of the permanently open files has no effect and generates no error.

CLOSE MULTIPLE FILES CAOP 18 12H

PURPOSE: Close a group of files.

REGISTERS ON ENTRY:

A contains 255 if all open files are to be closed (except the permanently open files mentioned in the CLOSE notes). Otherwise, A contains the number of the unit on which all files are to be closed.

PROCEDURE:

A CLOSE operation is performed for each file in the category determined by the value in register A. Device files are not closed unless that value is 255. Because there is no check on the value in register A, an attempt to close files on a nonexistent unit will have no effect.

No level 1 or 2 errors

CONTROL/STATUS CTLOP 9 13H

PURPOSE: Allow non-standard operations to be performed on a device.

REGISTERS ON ENTRY:

A contains the number of the device file.
B, DE, and HL contain parameters that are passed to the device driver.

PROCEDURE:

The Control/Status entry to the device driver is taken. The values in registers other than A are recognized by the specified driver, and the driver responds accordingly. Values may be returned to the calling program in registers A, DE, and HL.

Level 1 errors:	PTDEFS name:	Error codes: HEX	DEC
ILLEGAL DRIVER ACCESS	ERIDA	16	22
ILLEGAL FILE NUMBER	ERIFI	32	50

errors produced by device driver

NOTES:

Section 9 of this manual contains detailed information about the structure of device drivers; it also outlines recommended conventions, operation codes, and error returns.

CREATE CREOP Ø ØH

PURPOSE: Create a new file.

REGISTERS ON ENTRY:

DE points to a block that has the format:

#Bytes	Description
1	file type
2	block size
1	protection attributes
up to 1Ø	file name

PROCEDURE:

The directory is searched on the unit specified in the file name. No file with the specified name may exist on the diskette, and the directory must have room for the new entry. A file identification number (File ID) is assigned, the file is entered in the directory, and one disk block is allocated to the new file.

Level 1 errors:	PTDEFS name:	Error codes: HEX	DEC
FILE ALREADY EXISTS	ERAEX	2	2
ILLEGAL NAME	ERINM	ØD	13
ILLEGAL CHARACTER IN UNIT	ERICU	ØE	14
DISK FULL	ERDFL	1D	29
DIRECTORY FULL	ERDIR	1E	3Ø
SYSTEM WRITE-LOCKED	ERLOC	28	4Ø

Level Ø errors:

NO FILE ID'S	ERNID	1C	28

No level 2 errors

NOTES:

File type, protection attributes, and block size are discussed in Section 5.

No file will be created if there is an error return.

DELIMITED READ DRDOP 21 15H
DELIMITED WRITE DWROP 22 16H

PURPOSE: Transfer data exactly as in READ and WRITE (see below) but terminate the transfer if a specified character is encountered.

REGISTERS ON ENTRY:

A contains the number of the file.
BC contains the maximum number of bytes to be transferred.
DE contains the memory address from or to which the bytes are to be transferred.
L contains the delimiter.

PROCEDURE:

These operations are like READ and WRITE, with one added feature: it is possible to specify a character that, if encountered, will cause the READ or WRITE to be discontinued. Thus, the transfer of bytes will stop if 1) the count in BC reaches zero, 2) the end of the file is encountered, or 3) the character passed in register L is encountered. In the third case, the delimiter is the last byte to be transferred, and control returns to the normal return point with the registers altered as in READ and WRITE.

NOTES:

Error conditions and notes are identical to those for READ and WRITE.

DELIMITED SPACE DSPOP 23 17H

PURPOSE: Move the file cursor just beyond a specified character.

REGISTERS ON ENTRY:

A contains the file number.
BC contains the maximum distance that the cursor should be moved.
D specifies the type of cursor movement.
L contains the delimiter.

PROCEDURE:

If a change has been made in the block that is in the buffer and the cursor is moved to another block, the buffer is flushed to the diskette. The cursor is moved in the direction specified by the value in the D register.

The following values of D are legal:

D=Ø Rewind (as in SPACE)

D=-1 Space to end of file (as in SPACE)

D<128 but not zero (typically, D = 1)

Space forward a maximum of BC bytes. This procedure is similar to SPACE, except that spacing discontinues not only if BC = Ø or the end of the file is reached, but also if the delimiting character supplied in L is encountered. If the delimiter is encountered, the cursor will point to the byte following the delimiter. The registers are updated, as in SPACE.

D>127, but not -1 (typically, D = 128)

Space backwards a maximum of BC bytes. The file cursor is moved toward the beginning of the file, as in SPACE, except that spacing discontinues not only if BC = Ø or if the beginning of the file is reached, but also if the delimiting character supplied in L is encountered. If the delimiter is encountered, the cursor will be left pointing to the byte following the delimiter. The registers are updated, as in SPACE.

Errors are identical to those for SPACE.

ENDFILE EOFOP 6 6H

PURPOSE: The current position of the file cursor becomes the end of the file, and all subsequent data are lost.

REGISTERS ON ENTRY:

A contains the file number.

PROCEDURE:

The file must not be write-protected. The End-File entry to the appropriate driver is taken. If the file to be endfiled is a disk file, the current block (with the new EOF) is written to the

diskette, and any subsequent blocks are released. If the file has an index block for random access, that index is updated, i.e., the system automatically performs a RANDOM operation. The cursor is left pointing one position beyond the last byte of data.

Level 1 errors:	PTDEFS name:	Error codes HEX	DEC
ILLEGAL FILE NUMBER	ERIFI	32	50
PROTECTED FILE	ERPRO	4	4
FILE OPEN	ERMOP	6	6
NO SPACE FOR BUFFER	ERMOV	1B	27
ILLEGAL FILE NUMBER	ERIFI	32	50

No level 2 errors

NOTES:

Level 0 (very serious) errors that occur during this operation can result in the loss of some disk space formerly used by the file. To recover the space (but not the data it contained), make a copy of the file, KILL the original, and run the RECOVER program described in Section 2.2.

This operation cannot be performed on a file that is multiply open.

FINFO INFOP 11 0BH

PURPOSE: Provide information about a file.

REGISTERS ON ENTRY:

DE points to the file name
HL points to the area of memory that will receive the information about the specified file.

PROCEDURE:

The directory is searched on the unit specified in the file name. If the file exists and does not have the information-protect attribute, the information is stored at the address in HL, as illustrated below.

ON RETURN:
On return from the FINFO operation, the system will have stored the following information beginning at the address to which the HL register pair points at the time of the call.

# Bytes	Description	Notes
2	file ID,	
2	pointer to index block, sector	value is Ø if no index block
	" " " " track	
2	pointer to first block of file, sector	
	" " " " " " track	
2	number of blocks in file	
1	reserved for future use	
1	open flag	value of 1 if file is open; else Ø
1	reserved for future use	
1	file type	
2	block size	
1	protection attributes	
8	file name	zero-filled on right
1	Ø	
24		

Level 1 errors:	PTDEFS name:	Error codes: HEX	DEC
MEMORY PROTECT	ERSMP	12	18
Level 2 errors:			
FILE DOES NOT EXIST	ERNEX	1	1

KILL KILOP 2 2H

PURPOSE: Destroy a file and recover disk space used by the file.

REGISTERS ON ENTRY:

DE points to the name of the file to be killed.

PROCEDURE:

The directory is searched on the unit specified in the file name. In order to be KILLed, the file must exist and must not be open or have the kill-protect attribute. A buffer with length equal to the block size of the file is allocated in the system buffer area. The directory entry for the file is removed, and the diskette space occupied by the file is released for other use. Finally, the buffer is released.

Level 1 errors:	PTDEFS name:	Error codes: HEX	DEC
FILE DOES NOT EXIST	ERNEX	1	1
FILE PROTECTED	ERPRO	4	4
FILE ALREADY OPEN	ERMOP	6	6
NO SPACE FOR BUFFER	ERMOV	1B	27

No level 2 errors

NOTES:

Level Ø (very serious) errors may result in the loss of the diskette space occupied by the file. Make a copy of the file, KILL the original, and run the RECOVER program (see Section 2.2) to reclaim the lost space.

OPEN OPEOP 1 1H

PURPOSE: Create a data access path between the file and the calling program.

REGISTERS ON ENTRY:

DE points to the name of the file.
HL contains one of the buffer options:

-1	means dynamic buffering
Ø	means static buffering
other	is the beginning address for a user buffer

PROCEDURE:

The specified file must exist in the directory on the unit specified in the file name. If a static system buffer is selected, that is, if HL = Ø, a buffer is allocated in the system-managed buffer area. If a user buffer is specified, that is, if the value in HL is neither Ø nor -1, that buffer must not overlap the system in any way, or an error is generated. Addresses below GLPRO (user memory protect) are protected likewise.

If the named file is a device file (type = 255), the procedure is as follows:

1) The file is read; its contents are interpreted as an image file and loaded into memory. Only the first segment is loaded, and the starting address, if there is one, is ignored. The buffer address or option in HL is used for this load.

2) The buffer is released and a new one whose size is specified in the driver table (see Section 9) is allocated as described above.

3) The Initialize entry to the driver is called.

4) Prior to return, registers DE are set equal to the size of the driver just loaded. Registers HL are set to the address at which the driver was loaded.

It is recommended that dynamic buffering be avoided when opening a device file.

Whether the file is a disk or device file, the file cursor is set to point to the first byte of the file, and control returns to the calling program with the file number in register A and the file type in register B.

Level 1 errors:	PTDEFS name:	Error codes: HEX	DEC
FILE DOES NOT EXIST	ERNEX	1	1
BAD DEVICE FILE	ERZBC	9	9
ILLEGAL BUFFER ADDRESS	ERIBF	ØB	11
ILLEGAL NAME	ERINM	ØD	13
MEMORY PROTECT	ERSMP	12	18
TOO MANY FILES OPEN	ERTOP	1A	26
NO SPACE FOR BUFFER	ERMOV	1B	27

No level 2 errors

NOTES:

If a file is opened more than once, it may not be shortened by any of its openers; it may be extended only by the first opener.

Section 5.6 contains a detailed description of file access and buffering.

RANDOM RNDOP 17 11H

PURPOSE: Create or update the index of a random file.

REGISTERS ON ENTRY:

A contains the file number.

PROCEDURE:

This call is legal only for disk files. The current buffer is written out if a change was made in its contents. A 256-byte index block is allocated and attached to the file, if the file did not already have one. The file is rewound and then read sequentially; during this read, an index is constructed for no more than the first 128 blocks of the file. (Thus, no file may be accessed randomly beyond its 128th block.) If the return is normal, the cursor points one position beyond the last byte of the file, i.e., to the end of the file. The index block is written out, and its address is recorded in the directory entry for the file.

Level 1 errors:	PTDEFS name:	Error codes: HEX	DEC
FILE DEVICE TYPE	ERRDV	8	8
DISK FULL	ERDFL	1D	29
NO SPACE FOR BUFFER	ERMOV	1B	27
ILLEGAL FILE NUMBER	ERIFI	32	50

No level 2 errors

READ	RBLOP	3	3H
WRITE	WBLOP	4	4H

PURPOSE: Transfer data between a file and a selected area of memory.

REGISTERS ON ENTRY:

A contains the file number.
BC contains the number of bytes to be transferred.
DE contains the address to or from which data are to be transferred.

PROCEDURE:

The specified file may not have attributes that protect it against the operation to be performed, i.e., reading or writing. A read may not overwrite the system or user-protected areas. The transfer of data begins at the current cursor position. As data are transferred, the cursor is advanced through the file; registers DE are incremented, and registers BC decremented, for each byte transferred. At the end of the operation, DE will point one position beyond the last byte transferred, and BC will have been decremented by the number of bytes transferred (that is, BC will contain a value of zero if the return is normal.) Buffers are emptied or filled as the transfer progresses.

The transfer is terminated if BC reaches zero, if a serious error occurs, or if the end of the file is reached (in the case of READ). The only normal return is that in which BC has reached zero. If BC is equal to zero upon entry to READ or WRITE, no data are transferred.

Writing beyond the previous end of a file may cause additional blocks of storage to be allocated and added to the file, unless one of its attributes is Disk Allocation Protect. Also, if the file is multiply open, only the first opener can extend the file. (None can shorten the file.)

Level 1 errors:	PTDEFS name:	Error codes: HEX	DEC
FILE PROTECTED	ERPRO	4	4
MEMORY PROTECT	ERSMP	12	18
DISK FULL	ERDFL	1D	29
ILLEGAL FILE NUMBER	ERIFI	32	50

errors produced by device driver

Level 2 errors:

END OF FILE	EREOF	18	24

NOTES:

Values are returned in the registers as described above, regardless of errors. Some level Ø (very serious) errors (e.g., DISK FULL) will prevent the last buffer from being transferred successfully; data may be lost as a result.

RESET SREOP and RESOP 2Ø 14H

PURPOSE: Return to the system, but less drastically than if ABORT were called.

REGISTERS ON ENTRY:

No values are required.

PROCEDURE:

The system stack is made current, and files #Ø and #1 are made the CI input and output files, respectively. The buffer for file #Ø is cleared, so that no partial or complete commands will be executed when the Command Interpreter gains control. The prompt is reset to asterisk (*), the system utility file is made current, and the CI return trap (GLTRP) is disabled. Error traps (GLERH, GLERM and GLERS) and interrupt routine addresses (GLBD1, GLBD2) are set to their standard values, and control is passed to the CI. No error messages are printed, and no files are closed.

There are no error returns from this call.

NOTES: Section 3.3.2 contains more detailed information.

RETURN RETOP 13 ØDH

PURPOSE: Return control to the CI, enabling it to continue normal command processing.

REGISTERS ON ENTRY:

No values are required.

PROCEDURE:

The system stack is made current, and the error trap addresses (GLERH, GLERM, and GLERS) are set to their standard values. The interrupt routine addresses (GLBD1, GLBD2) are set to disable interrupt handling. The CI return trap address (GLTRP) is examined; if it is not equal to -1, control is transferred to the address given, after GLTRP is reset to -1. If GLTRP is equal to -1, the CI gains control. In either case, control will not return to the calling program.

There are no error returns from this call.

NOTES: See Section 3 for a more detailed discussion.

RETURN AND SET TRAP RTROP 24 18H

PURPOSE: Return control to the CI and execute the next command as in RETURN, but set the CI return trap word to the specified address, so that the next RETURN operation will return to that address, rather than to the CI.

REGISTERS ON ENTRY:

HL contains the address of the return trap.
If HL = -1, returns will not be trapped.

PROCEDURE:

The procedure for this call is identical to that for RETURN (see above), except that whereas RETURN sets GLTRP to -1 before transferring control to the current trap address, RETURN AND SET TRAP sets that parameter to the value passed in the HL register pair.

NOTES: This call makes it possible for a program to execute commands. The commands used must not alter the calling program, i.e., they must not use memory used by the caller.

There are no returns from this call, except indirectly through the CI return trap address.

SEEK SEKOP 16 10H

PURPOSE: Position a file cursor to a specified byte or the beginning of a specified block of a file.

REGISTERS ON ENTRY:

A contains the file number.
B contains the positioning option (byte if zero, block if not zero)
HL has the number of the byte or block at which the cursor
   will be positioned. (H = 0 for block positioning.)

PROCEDURE:

The current block is written to the disk if 1) a write operation has changed the current block since it was read, and 2) the SEEK operation entails moving to a new block. Then the SEEK entry to the appropriate device driver is taken.

If the file is a disk file, SEEK consults the index block and locates the block containing the desired byte. That block is read into the buffer for the file, and the cursor is set to point to the desired byte. (This byte must be within the bounds of the file; otherwise, an error occurs.) The 16-bit length of HL restricts positioning of the cursor to the first 65536 bytes of the file. Block positioning may be used to access bytes beyond byte 65535.

The SEEK operation cannot be performed on a file that does not have an index block (created by RANDOM). The CONTROL/STATUS operation may be used to load the index into user memory; this procedure makes random access more efficient by eliminating the need for an index load operation each time a SEEK is performed. If the index is loaded into memory with CONTROL/STATUS, SEEK will not need to access the disk more than once in order to load the desired block. (If the desired block is already in the buffer, SEEK will not have to access the disk at all.)

Level 1 errors:	PTDEFS name:	Error codes: HEX	DEC
FILE NOT RANDOM	ERRAC	7	7
ADDRESS OUT OF RANGE	ERSEK	19	25
NO SPACE FOR BUFFER	ERMOV	1B	27
ILLEGAL FILE NUMBER	ERIFI	32	50

errors produced by device driver

No level 2 errors

NOTES: If the ADDRESS OUT OF RANGE error is returned, the file cursor is left at an undefined position. Other errors leave the cursor unchanged.

SET UNIT SUNOP 12 ØCH

PURPOSE: Select a particular unit as the default unit (see name resolution in the introduction to this section).

REGISTERS ON ENTRY:

A contains the number of the new default unit.

PROCEDURE:

A must be in the range Ø to GLMXU-1 (GLMXU is a parameter in the System Global Area). If the value supplied in A is legal, the new default unit number is put into GLUNI in the System Global Area. Thereafter, any file named without a unit specifier will be assumed to exist on the new default unit.

Level 1 errors:	PTDEFS name:	Error codes: HEX	DEC
ILLEGAL UNIT	ERIUN	ØF	15

No level 2 errors

SPACE SPAOP 5 5H

PURPOSE: Move the file cursor relative to its current position.

REGISTERS ON ENTRY:

A contains the file number.
BC contains the distance that the cursor is to be moved.
D specifies the type of cursor movement.

PROCEDURE:

The current block is written out if 1) a WRITE operation has changed the current block since it was read and 2) the cursor is being moved to another block. Then the cursor is moved; the value in the D register dictates in which direction the cursor is to move, and the BC register pair specifies how far.

The possible values for register "D" are:

D=Ø The file is "rewound". BC is ignored. The Rewind entry into the device driver associated with the file is taken. For disk files, this causes the first block of the file to be read into the buffer and the file cursor to be set to point to the first byte of the file.

D=-1 The cursor is set to point to the End-of-File (EOF). BC is ignored. The Read Next Block entry into the device driver is taken repeatedly until an EOF condition is encountered. The file cursor is set to point past the last byte of data in the file.

D<128 but not Ø (typically, D = 1). Space forward BC bytes. The cursor is advanced toward the EOF by BC bytes, if possible. The buffer is loaded with the next file block when necessary. As the movement proceeds, BC is decremented by one for each byte passed. The movement of the cursor terminates when BC = Ø (normal return), when an EOF is encountered, or when some more serious error occurs.

D>127 but not -1 (typically, D = 128). Space backward BC bytes (toward the beginning of the file). Buffers are loaded using the Read-Last-Block call to the driver The movement of the cursor terminates when BC = Ø (normal return), the beginning of the file is reached, or a serious error occurs.

Level 1 errors:	PTDEFS name:	Error codes: HEX	DEC
ILLEGAL DRIVER ACCESS	ERIDA	16	22
NO SPACE FOR BUFFER	ERMOV	1B	27
ILLEGAL FILE NUMBER	ERIFI	32	5Ø

errors produced by device

Level 2 errors:

END OF FILE	EREOF	18	24

SECTION 8

SYSTEM UTILITIES

8.1 INTRODUCTION

The PTDOS System Diskette contains a utility file named SUTIL. This file is permanently open; it is file #2. When PTDOS is bootstrapped, the system global parameter GLUTF is set to 2, making SUTIL the current utility file (see Section 6). It is possible for the user to reset GLUTF to make some other utility file current, but when errors occur or when RESET or ABORT system calls are made, the system resets GLUTF to make SUTIL the current utility file again. This is done because SUTIL provides the mechanism used by the system to generate error messages. SUTIL contains four modules: the Explain Error Utility, the two data sets containing the error messages, and the File Catalog Utility.

8.2 EXPLAIN ERROR UTILITY (UXOP)

Module Ø, given the name UXOP in PTDEFS, is a program that displays error messages on the system console. It is used by the system to explain errors and can also be invoked by a user program. Modules 1 and 2 contain the text of the messages.

8

UXOP has the following calling sequence:

```
MVI A,<control word>
CALL  UTIL
DB  UXOP              (equated to Ø in PTDEFS)
JMP <error routine>
DB <operation code>
DB <error code>
<optional normal return point>
```

On entry, register A contains a control word that has the following format:

Bit 7:	Ø	call RESET after message
	1	return to calling routine's normal return point
Bit 3-6:	X	ignored
Bit 2:	Ø	CRLF after message
	1	no CRLF after message
Bit 1:	Ø	no "CALLED FROM" message on second line
	1	output "CALLED FROM" message on second line
Bit Ø:	Ø	HL points to a string (terminated by a zero byte) to be output as a second line
	1	HL contains a 16-bit quantity to be output on the second line
	X	ignored if HL=-1 (FFFFH)

The first line of an error message produced by UXOP has the format:

<operation> ERROR: <message>

The operation code following the UTIL call determines what operation name will appear in the message. The codes recognized are those corresponding to PTDOS system calls (see Section 7); if the operation code is -1 (FFFFH), no operation name will be output.

The error code determines what message will follow the colon; if the error code is -1 (FFFFH), nothing will be output following the colon. Error codes from 1 to 29H and 39H may be returned from system calls. The messages with codes between 2AH and 38H are provided for use by user programs; some of them are used by system commands. All of the error codes are assigned symbolic names in PTDEFS. The following table lists the symbolic names and messages corresponding to the error codes recognized by UXOP. Their standard meanings are discussed in Section 10.

NAME	ERROR CODE Hex	Dec	MESSAGE
ERNEX	1	1	FILE DOES NOT EXIST
ERAEX	2	2	FILE ALREADY EXISTS
ERUFN	3	3	FILE NOT OPEN
ERPRO	4	4	FILE PROTECTED
ERNIF	5	5	FILE NOT IMAGE TYPE
ERMOP	6	6	FILE ALREADY OPEN
ERRAC	7	7	FILE NOT RANDOM
ERRDV	8	8	FILE DEVICE TYPE
ERZBC	9	9	BAD DEVICE FILE
ERIOP	A	10	ILLEGAL OPERATION
ERIBF	B	11	ILLEGAL BUFFER ADDRESS
ERIBS	C	12	ILLEGAL BLOCK SIZE
ERINM	D	13	ILLEGAL NAME
ERICU	E	14	ILLEGAL CHARACTER IN UNIT
ERIUN	F	15	ILLEGAL UNIT
ERNTL	10	16	NAME TOO LONG
ERMEM	11	17	USER MEMORY PROTECT
ERSMP	12	18	MEMORY PROTECT
ERUCN	13	19	UNIT CONFLICT
ERIUP	14	20	ILLEGAL OPERATION
ERDRI	15	21	DRIVER ERROR
ERIDA	16	22	ILLEGAL DRIVER ACCESS
ERNCT	17	23	NON-RESPONDING DRIVER
EREOF	18	24	END OF FILE
ERSEK	19	25	ADDRESS OUT OF RANGE
ERTOP	1A	26	TOO MANY FILES OPEN
ERMOV	1B	27	NO SPACE FOR BUFFER
ERNID	1C	28	NO FILE ID'S
ERDFL	1D	29	DISK FULL
ERDIR	1E	30	DIRECTORY FULL
ERBLF	1F	31	BAD IMAGE FILE
ERFSB	20	32	BAD FILE STRUCTURE
ERFIC	21	33	FILE ID CONFLICT
ERBSC	22	34	BLOCK SIZE CONFLICT
ERSCC	23	35	SECTOR CONFLICT
ERCFS	24	36	CAN'T FIND SECTOR
ERTRI	25	37	BAD DISK STRUCTURE
ERCFT	26	38	CAN'T FIND TRACK
ERRBC	27	39	READ-BACK FAILED
ERLOC	28	40	SYSTEM WRITE-LOCKED
ERXXX	29	41	CATASTROPHIC ERROR
ERSYN	2A	42	COMMAND SYNTAX ERROR
ERNAX	2B	43	EXPECTED A NAME
ERNUX	2C	44	EXPECTED A SYMBOL
ERBCX	2D	45	EXPECTED A BYTE COUNT
ERADX	2E	46	EXPECTED AN ADDRESS
ERVAX	2F	47	EXPECTED A VALUE
EROPX	30	48	EXPECTED AN OPTION PARAMETER
ERINA	31	49	ILLEGAL FILE NAME
ERIFI	32	50	ILLEGAL FILE NUMBER
ERIAD	33	51	ILLEGAL ADDRESS
ERIBC	34	52	ILLEGAL BYTE COUNT
ERIVA	35	53	ILLEGAL VALUE
ERIOS	36	54	ILLEGAL OPTION SPECIFIER
ERITY	37	55	ILLEGAL TYPE
ERIAT	38	56	ILLEGAL ATTRIBUTES

8.3 FILE CATALOG UTILITY (UCAT)

Segment 3, given the name UCAT in PTDEFS, is a program that can be invoked to obtain directory information for files of a specified type on a specified disk unit. It returns the name, length in sectors, and protection attributes for each qualifying file in a buffer area allocated by the calling program. For example, the BASIC CAT command, which displays a listing of BASIC program or data files, obtains the information to be displayed by means of a call to UCAT.

UCAT has the following calling sequence:

```
MVI A,<file type>
LXI H,<address of unit number/data buffer>
CALL UTIL
DB UCAT                 (equated to 3 in PTDEFS)
JMP <error routine>
<normal return point>
```

On entry, register A contains the binary file type for which information is desired. Register pair HL points to a memory location that contains the binary disk unit number to be searched. Immediately following this location, a buffer area must be allocated in which the information will be returned. The requisite length of this buffer area is 12*NFILES+1, where NFILES is the maximum number of qualifying files.

On return, register A contains the number of qualifying files. Register pair HL points to the last memory location actually used in the buffer (i.e. the terminating zero byte). The buffer contains one 12-byte entry for each qualifying file; the last entry is followed by a zero byte to mark the end of the data. The file information entries have the following format:

#Bytes	Description
8	File Name (ASCII, filled on right with NULs)
3	Length in Sectors (decimal ASCII, leading zeros)
1	Protection Attributes (see Section 5.2.3)

SECTION 9

DEVICE DRIVERS

9.1 INTRODUCTION

A device driver is a program that controls a hardware device such as a line printer, cassette tape drive, paper tape reader or punch, magnetic tape drive, etc. PTDOS has a "device independent" I/O system; device drivers take the form of "device files" that look to the system exactly like disk files, with the exception that some operations (e.g., random access operations) may not be meaningful on a particular device.

This section describes the format of device files, so that a user may interface special hardware to PTDOS. The standard device drivers for the system console and CUTS format cassette tape are also discussed.

9.2 DEVICE FILE FORMAT

A device file is an image file with file type 255 (FFH, or symbolic type 'D'.) It may have only a single segment (see Section 5.7.3). The load address for this block is the address at which the driver will be loaded by PTDOS when the device file is accessed. The first 25 bytes comprise a "driver table", which is the interface between PTDOS and the driver; it contains information about the driver, and specifies the entry point addresses for routines that implement a standard set of device functions for the particular hardware peripheral. The driver table is followed by the code for these routines.

9.2.1 Driver Table Format

The driver table has the following format. Each 2-byte entry is stored with the low-order byte first. If a particular operation is not supported by a driver, the corresponding address in the table should be zero. Then, if an attempt is made to perform that operation, an ILLEGAL DRIVER ACCESS error (level 1) will be generated by the system.

Any errors detected by the driver should be handled by calling the ERRLØ, ERRL1, or ERRL2 system entry point with the appropriate error code (see Section 6.4.3).

CONTENTS OF THE DRIVER TABLE

NAME OF ENTRY	DESCRIPTION	#BYTES	KIND OF ENTRY	#RETURNS
DTRB	Read block	2	Address	2
DTRNB	Read next block	2	Address	2
DTRLB	Read last block	2	Address	2
DTWBR	Write block RN	2	Address	2
DTWB	Write block	2	Address	2
DTREW	Rewind	2	Address	1
DTEOF	Endfile	2	Address	1
DTCLO	Close	2	Address	1
DTSEK	Seek	2	Address	2
DTCTL	Control/status	2	Address	2
DTBLK	Block size	2	Data	-
DTITO	ITO	1	Flag	-
DTINI	Initialization	2	Address	1

9.2.2 Calling Sequences for Driver Routines

The calling sequences for the various driver routines are described below. In the descriptions:

Entry: refers to the values to be expected in the registers when the particular driver routine is entered.

Return1: refers to a return performed by a RET instruction (or equivalent).

Return2: refers to a return performed by first incrementing the address on the top of the stack three times, and then then performing a RET instruction (or equivalent).

NAME OF ROUTINE	FUNCTION AND CALLING SEQUENCE

DTRB — Read Block

Used for loading buffers in cases other than those listed below (DTRNB, DTRLB).

```
Entry:    HL = buffer address (data destination)
          DE = maximum number of bytes to read
Returnl:  EOF encountered
Return2:  Normal return
          HL = number of bytes read.
```

DTRNB — Read Next Block

Used by the READ and RB operations.

Calling sequence is identical to DTRB.

DTRLB — Read Last Block

Used by the SPACE operation to read the previous block while spacing backwards.

Calling sequence is identical to DTRB except that Returnl is taken if the beginning of the file (BOF) is encountered.

DTWBR — Write Block; Read Next Block

Used by the WRITE and WB operations. A sequential device driver should return a zero in HL.

```
Entry:    HL = buffer address (data source)
          DE = number of bytes to transfer
Returnl:  EOF encountered reading next block
Return2:  Normal return
          HL = number of bytes read
```

DTWB — Write Block

```
Entry:    HL = buffer address (data source)
          DE = number of bytes to transfer
Returnl:  EOF encountered (e.g., end of tape)
Return2:  Normal return
```

DTREW — Rewind

Used by the SPACE operation. A sequential device device driver should return a zero in HL.

```
Entry:    HL = buffer address
          DE = buffer size
Returnl:  Normal return
          HL = number of bytes read
```

DTEOF — Endfile

Used by the ENDFILE operation.

Entry: HL = buffer address (data source)
DE = buffer size
Returnl: Normal return

DTCLO — Close

Used by the CLOSE operation. Once the driver has been closed, it must be reopened before further use.

Entry: No parameters
Returnl: Normal return

DTSEK — Seek

Used by the SEEK operation.

Entry: HL = buffer address (data destination)
A = value in B when SEEK call was made
DE = buffer size

The 16-bit seek address is passed on the stack immediately under the return address. The following code must be executed to remove it from the stack before a return is made (even if, for some reason, it is not needed by the driver routine).

```
POP  H
XTHL    seek address is now in HL
```

Returnl: Seek address out of range
Return2: Normal return
HL = number of bytes loaded into buffer
DE = displacement into the block where the file cursor is positioned

DTCTL — Driver Control/Status Request

Used by the CONTROL/STATUS operation.

Entry: A = value in B when CONTROL/STATUS system call was made
B = operation code (see list below)
DE = value when CONTROL/STATUS system call was made
HL = value when CONTROL/STATUS system call was made

Returnl: Normal return
A, DE, and HL are returned to the program that made the CONTROL/STATUS system call.

If the driver does not support a requested control/status operation, it should call the ERRL2 system entry point with an error code of 17H (ERNCT).

The operation codes for control/status requests are defined as follows:

CODE — OPERATION

Ø Return Driver Status

Returnl: A = protection attributes
D = device characteristics, format is

Bit	Meaning
7	Echoes input
6	Interactive
5	Handles tab characters
4	Tab stops may be set
3	Handles ASCII formfeed
2	Can do absolute tabs
1	Can be interrupt driven
Ø	Supports set prompt operation

1 Forms Control

The function to be performed is determined by the value value in register E, as follows:

Value	Function
Ø	Do a formfeed.
1	Set tab stop at current position.
2	Clear tab stop at current position.
3	Clear all tab stops.
4	Absolute tab to position in HL.

2 Set Prompt

Input prompt is set to the string to which register pair HL points. The string must be terminated by a NUL (ØØH).

3 Reset Device

4 Load Random Index

This operation is meaningful only for disk files. The index block is loaded at the address in HL. Calling subsequently with HL = -1 (FFFFH) informs the driver that the index is no longer in memory.

5 Turn On Echo

6 Turn Off Echo

7 Special Status Read

Returnl: A = device ready status (Ø if not ready)
DE = number of bytes of free space in the driver's buffer (for disk files, free space in the PTDOS buffer)

8 Set device-not-ready trap address

Entry: HL = address of routine to be called if device is not ready (Ø clears a previously set trap address)

A trap routine runs on the driver's stack unless it switches to its own while executing; return to the driver is effected with a RET instruction or equivalent. A trap routine may not make any system calls except to CONIN, CONOUT, and CONTST. Interrupts are disabled before the trap routine is entered.

The disk driver behaves in the following way when a drive is found not to be ready. If no trap address has been set, the disk driver displays the message DRIVE NOT READY on the system console and waits for a key to be typed. If the key was a carriage return, another attempt will be made to access the disk; if the key was a mode select, the Command Interpreter will be re-entered. If a trap address has been set, the trap routine is called. The zero flag is cleared upon return from the trap routine, that routine will be called again; if the zero flag is set, the driver will call the ERRL1 system entry point with an error code of 39H (ERRED). The Command Interpreter, whenever it is re-entered, clears the disk driver trap address.

9 Refresh buffer

The driver's buffer (the PTDOS buffer for the disk driver) is loaded from the device.

1Ø Refresh device

The data in the driver's buffer (the PTDOS buffer for the disk driver) is written to the device.

11-31 Reserved for future system definition

32-255 User-definable

DTBLK Block Size

This is the block size for the device or the maximum block size for a device whose block size is variable (e.g., the the console device).

DTITO Immediate Transfer Option

If this entry is not zero, the buffer will be written to the device at the conclusion of all WRITE and WB operations.

DTINI Open-time Initialization

Used by the OPEN operation.

Entry: No parameters
Returnl: Normal return

9.3 BUILDING A DEVICE FILE

A device file may be created by the following procedure:

1) Write the driver table and assembly language program to implement the desired operations. The load address of the driver (set with an ORG pseudo-operation) must be chosen carefully, so that it will not conflict with any other drivers that may be used simultaneously. A start address (set with an XEQ pseudo-operation) is unnecessary and will be ignored if present.

2) Assemble the driver with ASSM.

3) Compress the resulting image file into a single logical block with EXTRACT. If the device file consists of more than one logical block, none except the first will be loaded when the device file is accessed.

4) Set the image file type to "D." with RETYPE.

5) Set the file protection attributes of the file with REATR. Normally, it is desirable to set Kill and Name protection. Remember that Read and Write protection apply to the operations that are allowed on the hardware device; Read and Write protection should be set only for a write-only or read-only device, respectively. Information protection prevents the device file from appearing in a FILES listing unless the command includes the S=-I argument; the GET and SAVE commands will also require this argument if they are to operate on information-protected files.

9.4 CONSOLE DRIVER

9.4.1 Device File Access

The system console is treated as two permanently open device files; console input is read from file #Ø and console output is directed to file #1.

An input line, terminated by a carriage return, constitutes a data block for file #Ø; its length may range from Ø to 8Ø characters, not including the CR. Each line is buffered by the driver as it is typed and is not returned by the driver until it is terminated by the CR. Until that time, single characters or the entire line may be deleted by typing the following special characters:

```
DEL     Delete one character
CTRL-X  Delete entire line;  ! CRLF is echoed
```

If a CTRL-C is entered as the first character of a line, it is interpreted as an end of file. The program reading data from the console must respond to this condition as it sees fit. For example, the Command Interpreter makes a CLOSE ALL system call when it encounters an EOF on the console input file.

The console input driver supports two CONTROL/STATUS operations: STATUS READ (B=Ø) and SET PROMPT (B=2). The console output driver supports the STATUS READ operation.

Some additional features of the console driver are controlled by three parameters in the system global area:

GLUPS If this parameter is set, lower case letters are upshifted on both input and output.

GLNCT The number of NULs to follow a LF on output.

GLBIO If this parameter is set, GLUPS and GLNCT are ignored and bit 7 of an input character is not masked to zero. This setting makes possible binary I/O through the console device, e.g., to and from a paper tape reader and punch.

9.4.2 Single Character Access

A character-at-a-time interface to the console driver is provided by the CONIN, CONTST, and CONOUT system entry points (see Section 6.4.4).

Console input and output through these entry points is affected by the global area parameters GLUPS, GLNCT, and GLBIO exactly as described above. Two additional system global parameters are useful for character-at-a-time console input:

GLBYT Holds the last character read.

GLFLG If the value of this parameter is not zero, the next character read will come from GLBYT rather than the console input device.

These parameters make it possible to put a character back into the input stream after it has been examined; the last character read will be read again if GLFLG is set to a non-zero value. Any other character may be forced into the input stream if it is put into GLBYT and GLFLG is set. Once the character in GLBYT has been used, GLFLG is reset to Ø by the system.

9.4.3 Hardware Interface

The interface between the console device driver and the hardware devices used for console input and output is accomplished by means of very simple routines that test for a waiting character, input a character, and output a character. PTDOS includes a set of these routines for keyboard input and VDM output, and for serial or parallel port input and output. When the system is initialized, the set of these routines appropriate for the hardware is chosen automatically.

The console device driver accesses the hardware interface routines through three addresses in the system global area:

GLTCH	pointer to the test for waiting character routine;
GLRCH	pointer to the read character routine;
GLWCH	pointer to the write character routine.

If the system is being run on either a Sol Terminal Computer (with the SOLOS monitor program) or some other machine with the CUTER monitor program resident at location C000H, the global area pointers are set in accordance with the current SOLOS/CUTER pseudo-port:

Pseudo-Port	GLRCH,GLTCH	GLWCH
0	keyboard	video display
1,3	serial port	serial port
2	parallel port	parallel port

The serial or parallel port routines, if used, are configured to match the hardware port assignments assumed by SOLOS or CUTER. The CUTER hardware port assignments are described in the SOLOS/CUTER User's Manual.

If neither SOLOS nor CUTER is found at location C000H, the CUTER serial port assignments are used:

PORT	DESCRIPTION
0	status port

Bit	Meaning
7	ready to transmit data, active high
6	received data available, active high
0-5	unused

1	data port

If non-standard hardware requires special interface routines, the pointers may be changed to the entry point addresses of those routines. Initially, the changes may be made in the appropriate system global area locations after bootstrapping; the new values of the pointers may be incorporated permanently into the system by changing the disk-based resident with the CONFIGR command (see Section 3). Unless the interface routines are resident in ROM, they must be loaded every time the system is bootstrapped. By making the routines into one or more image files and including their names, each

followed by a comma, in the START.UP file (see Section 2), the user can provide for automatic loading upon bootstrapping. (Because of the commas following their names, the routines will be loaded but not executed. See Section 2.)

If either GLRCH (and GLTCH) or GLWCH has been changed to point to a user-supplied interface routine, it will not be changed again as a result of the initialization procedure described above.

Interface routines must conform to the following specifications:

GLTCH	Return:	Zero flag is set if no character is available; otherwise, zero flag is cleared. Register A and the rest of the flags are undefined. All other registers must be unchanged.
GLRCH	Return:	Register A contains the character that was read; if GLBIO is zero, bit 7 must be masked to zero. The flag register is undefined. All other registers must be unchanged.
GLWCH	Entry:	Register B contains the character to be output.
	Return:	Registers A and B contain the character that was output. The flag register is undefined. All other registers must be unchanged.

9.5 CASSETTE TAPE DRIVER

A device driver for CUTS format cassette tape is included on the PTDOS system diskette. It enables cassette tape units interfaced to the Sol's audio cassette interface or to a Processor Technology CUTS board to be accessed as PTDOS files.

There are three files on the diskette:

CTAPE.A	assembly language source code;
CTAPE1	device file for tape unit 1, loads at 7000H;
CTAPE2	device file for tape unit 2, loads at 732BH.

The load addresses may be changed if desired by rebuilding CTAPE1 and/or CTAPE2 from the source code on CTAPE.A, following the procedure outlined in Section 9.3.

A PTDOS cassette file, as implemented by the CTAPE driver, is a SOLOS/CUTER multiple block file with a block length of 400H, file name 'PTFIL', and a non-executable file type of 'D'. The load address field in the block headers is used for a block sequence count. (See the SOLOS/CUTER User's Manual for a detailed description of the format.)

The CTAPE driver supports the following operations:

DTRB	reads a block from tape.
DTRNB	" " " " "
DTWB	writes a block to tape.
DTWBR	" " " " "
DTREW	turns on tape unit motor, displays rewind message.
DTEOF	writes an EOF block to tape.
DTCLO	writes an EOF block to tape if one has not yet been written.
DTSEK	spaces tape past next EOF block.
DTINI	simply returns, no initialization required.

9.6 NULL DEVICE FILE

There is a device file named NULL included on the PTDOS system diskette. It is intended to be used as a data sink in case a program requires an output file, but the user wishes to discard the output produced.

The NULL device driver supports all operations, but takes no action for any of the them. It loads at address 8000H.

SECTION 10

ERROR MESSAGES

PTDOS displays an error message if an error occurs during the execution of a system command or system call; a user program can display one of the same messages by calling the Explain Error Utility (UXOP) with the appropriate error code.

Sections 6 and 8 of this manual contain detailed information about error handling, the PTDOS Utility Handler, and the Explain Error Utility. This section is devoted to an explanation of the individual error messages. The messages are listed in alphabetical order; each message is followed by its decimal error code, its hexadecimal error code, an explanation of what might have caused the error, and some suggestions as to how the problem might be remedied. (Section 8 includes a list of the messages ordered by their error codes.)

If an error occurs during a system call, the code corresponding to that error is returned in the A register. THE ERRORS NUMBERED 2A THROUGH 38 IN THE LIST BELOW WILL NEVER OCCUR DURING A SYSTEM CALL, but the corresponding error messages are displayed by some PTDOS Commands. Any of the messages except DRIVE NOT READY may be displayed by a user program if a call is made to the Explain Error Utility.

DEALING WITH HARDWARE-RELATED ERRORS

The messages numbered 20H through 27H in the list below are normally associated with hardware problems, for example, a scratch on the disk, or faulty alignment of the disk drive. (In some cases, operations may be performed on a disk in one system but not in another, because the two disk drives are not compatible in their alignment.) Of the errors in this set, only BLOCK SIZE CONFLICT can result from an error in a user program. The proper course of action for dealing with any of these errors is:

1) Try the operation again; if the problem is minor, the second attempt might be successful. (Make a back-up copy of your file on another disk; even if the operation works the second time, the file on which you have been working is suspect.)

2) If the operation fails a second time, try to make a DISKCOPY of the disk that has given you trouble. Sometimes the copy you make of the bad file or files will not have the problem that the original file had.

3) Use the RECOVER command described in Section 3. ONLY USE THIS COMMAND IF YOU HAVE MADE A DISKCOPY OF THE BAD DISK; sometimes RECOVER can destroy information that you might otherwise be able to retrieve.

4) Consult Section 6 of the Helios II Disk Memory System User's Manual.

THESE ARE THE ERROR MESSAGES:

ADDRESS OUT OF RANGE 25 19H

This error arises when the SEEK command or system call is used to position the file cursor after the end of the designated file. If you were using the command, check your parameters. If you were making a system call, check the B register and the HL register pair; did you try to position the file cursor at the first byte of block 372, instead of at byte 372 of the file? This error causes the cursor to be left at an undefined position in the file.

BAD DEVICE FILE 9 9H

This error will occur if you try to open a device file less than 25 bytes long. (Not only the OPEN command, but also SETIN, SETOUT, COPY, PRINT, and other commands will cause a device file to be opened.) Make sure that you indicated the file you really wanted to open for input or output, and that the file contains a device driver. (Section 9 of this manual describes device files and their structure.)

BAD DISK STRUCTURE 37 25H

This message indicates that the disk drive hardware is unable to read the header of the first block of the file you are trying to access. Follow the suggestions under "DEALING WITH HARDWARE-RELATED ERRORS," above.

BAD FILE STRUCTURE 32 20H

This message means that the chaining pointers of two blocks on a diskette do not agree, that is, that the foreward chaining pointer for a given block A points to block B, but the backward chaining pointer for block B does not point to block A. This problem is hardware-related; follow the suggestions under "DEALING WITH HARDWARE-RELATED ERRORS," above.

BAD IMAGE FILE 31 1FH

The image file that you are trying to read contains fewer bytes than you are trying to read from it. (The structure of an image file is discussed in Section 5.5 of this manual.) Reassemble the program, if you have a copy of its source, or obtain another copy of the program.

BLOCK SIZE CONFLICT 34 22H

This error can be caused by a hardware problem affecting the directory or the named file, or by an error in the user's assembly language program; the message indicates that the block size recorded in the block header does not match the block size recorded in the directory or specified in the user program. Check your program for a possible error; otherwise, follow the suggestions under "DEALING WITH HARDWARE-RELATED ERRORS," above.

CAN'T FIND SECTOR 36 24H

This error is caused by a hardware problem; the message indicates that the disk drive hardware cannot find a header that identifies its location as being on the desired sector. Follow the suggestions under "DEALING WITH HARDWARE-RELATED ERRORS," above.

CAN'T FIND TRACK 38 26H

This error is similar to CAN'T FIND SECTOR; the disk drive hardware cannot find a header that identifies its location as being on the proper track. Follow the suggestions under "DEALING WITH HARDWARE-RELATED ERRORS," above.

CATASTROPHIC ERROR 41 29H

This error message usually means that it is impossible to bootload because something is wrong with the system diskette, or that you have performed some unusual or nonsensical operation. Try ejecting the diskette, inserting it again, and repeating the operation. (If the error did not occur during an attempt to bootload, try to bootload the system again.) If you are still unsuccessful, you should proceed very cautiously. Try to bootload with a diskette that is not very valuable to you or that is well backed up, and see whether a few commands work; if there are still problems, try putting the diskette in another disk drive unit. If you cannot operate with the other diskette, either, consult Section 6 of the Helios Disk Memory System User's Manual.

COMMAND SYNTAX ERROR 42 2AH

Some commands cause this message to be displayed if the command has not been entered correctly. Check the syntax of the command and enter it again.

DIRECTORY FULL 30 1EH

This message indicates that you have exceeded the maximum number of file names allowed in one directory. (The maximum number is 192.) If you do not want to put your file on a different diskette, you must either KILL a file that you no longer need, or concatenate two files (by appending one to another) so that you can kill the appended file without losing the information that it contained.

DISK FULL 29 1DH

You have filled up the diskette on which you are writing (with GET, COPY, etc.). Write your information to another diskette, or KILL old files to make more room on the full diskette. It is easy to avoid this error by giving the FREE? command occasionally. For example, if you are about to develop an EDIT file, give the FREE? command and specify the unit onto which you intend to write the new file. If you find out there are 435 sectors left on the diskette, you know that there is room for the new file; if there are only 10 sectors left, you had better consider creating your file on a different diskette (or killing some old files to make more room on the diskette that is nearly full).

DRIVE NOT READY 57 39H

This message is displayed by the driver and is not included in the Explain Error Utility; it almost always means that you have forgotten to insert a diskette into the drive, or that the diskette is upside down or backwards. Push the diskette all the way in, and then strike any key to initiate execution of the last command entered. If the diskette appears to be inserted completely (and right-side up), but you still get an error, push the eject button and insert the diskette again; then strike any key other than MODE SELECT to execute the last command. If you do not want the last command to be executed, strike the MODE SELECT (or CTRL and @ keys) to return to command mode. (If you choose to return to command mode, you do not need to insert the diskette before striking MODE SELECT.) This error can also be caused by any hardware problem that prevents a disk drive unit from reporting to the controller that it is ready; if you suspect a hardware problem, consult Section 6 of the Helios II Disk Memory System User's Manual.

DRIVER ERROR 21 15H

This message indicates an error return from a call to a device driver (for example, the keyboard or video display). Make sure that the device is properly connected.

END OF FILE 24 18H

This message means that you tried to access information before the beginning or after the end of a file. For example, the message might be displayed if you use the SPACE system call to position the cursor after the end of the file, or the READ system call to read more bytes than the file contains. If you were using SPACE, examine the BC register pair and the D register; did you request that the cursor be moved backward, instead of forward? If you were using READ, does the BC register pair contain too large a number?

EXPECTED A BYTE COUNT 45 2DH

This error indicates that a command required a byte count parameter and you did not supply one. Retype the command, specifying a byte count.

EXPECTED AN ADDRESS 46 2EH

This error indicates that a command required an address parameter and you did not supply one. Retype the command, specifying an address parameter.

EXPECTED A NAME 43 2BH

This error indicates that a command required a name parameter and you did not supply one. Retype the command, including a name. (You might get this error message if a command requires that you specify a file by its name, and you specify it by its number, instead.)

EXPECTED AN OPTION PARAMETER 48 30H

This error indicates that a command requires an option parameter (i.e., S=something) and you did not supply one. Retype the command, including an option parameter.

EXPECTED A SYMBOL 44 2CH

This error indicates that a program required a symbol and you did not supply one. Try again, specifying a legal symbol.

EXPECTED A VALUE 47 2FH

This error indicates that a program required a value and you did not supply one. Try again, specifying a legal value.

FILE ALREADY EXISTS 2 2H

This error will arise if you try to CREATE a file that already exists on the diskette. If you really meant to create a new file, change its name slightly so that it is no longer the same as that of the existing file.

FILE ALREADY OPEN 6 6H

This error results from an attempt to change a file that is also open for other operations that might be made impossible by that change. For example, if a file has been opened twice, as file numbers 5 and 6, endfiling file #6 would make nonsensical an attempt to SEEK to a later part of file #5. It is illegal to KILL or shorten a file that is open elsewhere in the system. Also, a multiply open file may have its length extended ONLY if that file is specified by the number assigned to it when it was first opened; of course, the file must still be open under that file number.

FILE DOES NOT EXIST 1 1H

This message indicates that you have attempted to load or operate on a file that does not exist on the diskette from which you are trying to read it. (Many programs, e.g., EDIT, DEBUG, will automaticaly create a file if the user specifies one that does not exist; otherwise, the user must create the file on the diskette before referring to it.) Be sure that you have not inserted the wrong diskette, and that you have specified a unit number if the file is on a diskette in a unit other than the default unit.

FILE ID CONFLICT 33 21H

This error arises when the foreward or backward chaining pointer of a given block A points to block B, but block B identifies itself as not belonging to the same file as block A; the message means that the file control block for the file does not match the directory or a sector on the diskette. Unless you opened a file on one diskette and tried to close it on another, this error is normally indicative of a hardware problem; follow the suggestions under "DEALING WITH HARDWARE-RELATED ERRORS," above.

FILE IS DEVICE TYPE 8 8H

The operation that you are trying to perform may not be performed on a device file. For example, you may be trying to randomize the file. Whatever command or call you are using, be sure that you have designated the file that you really want to access.

FILE NOT IMAGE TYPE 5 5H

This message indicates that you have attempted to load a file that is not an image file, probably by typing the name of that file after the PTDOS prompt (*). Type the name of the program that you actually intended to execute.

FILE NOT OPEN 3 3H

The number by which you have designated a file, either in a command or in the A register before a system call, does not correspond to the number of an open file. Make sure that 1) the file has been opened, but not closed, before you refer to it, 2) you have not mistaken the number assigned to the file when it was opened, and 3) you stored the file number where the program expected to find it, and did not write over it with anything else in the meantime. Remember that a file that is not open does not have a file number, and that it is therefore impossible to designate a file by number if that file is not open. Either open the file before you refer to it by number, or refer to the file by its name.

FILE NOT RANDOM 7 7H

This message indicates that you have tried to perform a random access operation on a file that has not been set up for random access. If you want to make the file a random access file, use the RANDOM command or system call to build the required index.

FILE PROTECTED 4 4H

This message indicates that you are trying to perform an operation on a file whose attributes protect it against that operation. For example, you may be trying to KILL a kill-protected file, or to RENAME a name-and-type-change-protected file. Unless a file is protected against a change in its attributes, you can use the REATR command to change the attribute that is hindering you. It is always a good idea to consider why a file has been given particular attributes; if you change the attributes of file so that you can perform a particular operation, remember to restore the old attributes at the conclusion of the operation.

ILLEGAL ADDRESS 51 33H

This message means that an address supplied to a program is somehow inappropriate, e.g., beyond the limits of addressable memory, inconsistent with the specifications of the program being executed, etc. If this message is displayed in response to a command line that you have typed on the console, type the command again, giving a suitable address specification.

ILLEGAL ATTRIBUTES 56 38H

This message means that you have attempted to give a file an attribute that is not in the list of possible attributes. Check the list in Section 5 of this manual. Then type the command again, assigning the correct attributes.

ILLEGAL BLOCK SIZE 12 ØCH

You have tried to assign a block size of Ø, or a block size greater than 4Ø95. If you were using the CREATE command, type it again with a legal block size parameter; if you were using the CREATE system call, change the block size bytes in the block of memory addressed by the DE register pair. Block size considerations are discussed in Section 5 of this manual.

ILLEGAL BUFFER ADDRESS 11 ØBH

This message indicates that you have tried to allocate a buffer in reserved memory, i.e., that the second parameter of the OPEN command, or the value in the HL register pair at the time of the OPEN system call, is either in user-protected memory or inside PTDOS. User-protected memory may be allocated by the SET PR= or CONFIGR

command; in either case, you can supply to the system a location below which you do not want programs or data to be loaded. The area occupied by PTDOS begins at GLLOW and ends at BFFFH. GLLOW has a default value of 9000H, but can be set lower with either the SET BU= or CONFIGR command. To deal with the ILLEGAL BUFFER ADDRESS message, either change the buffer address specification or configure the system so that the desired buffer address is legal. (Use SET to change system parameters in memory; use CONFIGR to change them on the diskette.)

NOTE:

It is impossible to establish a buffer at address 0 or FFFFH. A buffer address specification of 0 is taken to be not an address, but an indication that buffering should be static, and a specification of FFFFH (or 'T' in the OPEN command) is taken to indicate that buffering should be dynamic. A buffer in user memory is always static.

ILLEGAL BYTE COUNT 52 34H

This message indicates that a byte count supplied to a command or program does not meet the requirements of that program. Determine the range of byte counts acceptable to the program and proceed accordingly.

ILLEGAL CHARACTER IN UNIT 14 0EH

Unit numbers are specified as /u, where u is a number signifying the unit. (Remember that unit numbers begin at 0, not 1.) If a character other than a number appears after the slash, PTDOS will return this error message. Enter the command again with a legal unit number.

ILLEGAL DRIVER ACCESS 22 16H

This message is displayed when a driver is accessed for a purpose that the driver software is not designed to serve. (For example, it is usually not possible to rewind a printer.) Perhaps you gave the wrong file number in your command or program. This message does not necessarily mean that the operation you wanted to perform cannot be performed on any device; it means only that the operation is not appropriate for the particular device on which you have chosen to perform it.

ILLEGAL FILE NAME 49 31H

An illegal file name is a name that violates the rules imposed by a particular command or program. For example, a program may require that no file name be more than five characters long; than an attempt to use that program to create a file having a six-character filename might cause this message to be displayed. (The message NAME TOO LONG is also available for such cases, but in a situation where there might be several kinds of constraints on file names, ILLEGAL FILE NAME might be preferable on account of its generality.

ILLEGAL FILE NUMBER 50 32H

An illegal file number is a number that is less than 0 or greater than 255. If the error occurs during the execution of your own program, check to see where you specified the file number, and change it if it is incorrect. (If the file number seems to be correct, make sure that it is stored where your program expects to find it.) If you made a typing error when you entered a command, enter the command again with the correct file number.

ILLEGAL NAME 13 0DH

This message indicates that a file name supplied to a command contains an illegal character, i.e., one of the characters given in the list near the beginning of Section 5. If you were giving a command, enter it again with a legal file name.

ILLEGAL OPERATION 20 14H

This message means that you have specified an operation that is not one of the 25 available system calls (see Section 7). Usually this error results from a sequence like

```
CALL SYS
DB 26
```

in the user program. Because there are only 25 possibilities (numbered 0 to 24), the 26 does not signify an operation at all. Another possibility is that the statement after CALL SYS is not a DB pseudo-operation. The value stored by the assembler in the byte after CALL SYS will always be taken as data specifying the operation to be performed by the system. If an instruction follows the CALL SYS statement and its operation code is not greater than 24 Decimal, the corresponding system call will be made and no ILLEGAL PTDOS OP error will be reported (although some other error or dubious activity is certain to result); if the operation code is greater than 24 Decimal, an ILLEGAL PTDOS OP error will be reported.

ILLEGAL OPTION SPECIFIER 54 36H

This message is displayed if the letter after the S= in a command line is not the symbol for an available option. Check the syntax of the command, and enter it again with an appropriate option specifier.

ILLEGAL TYPE 55 37H

This message is displayed if you specify 1) any file type consisting of more than two characters, or 2) a non-image file type consisting of more than one character. See Section 5 of this manual for a discussion of file types.) Enter the command again, giving a legal type.

ILLEGAL UNIT 15 ØFH

This message is displayed if the number following the slash (in a unit designation of the form /u) exceeds the maximum number of units for which the system is configured. The CONFIGR command may be used to change the maximum number of units to be supported by the system; that parameter is set initially to 2. If you made a typing error, enter the command again with the proper unit number; if the system was configured incorrectly, use SET DU= to change the configuration in memory, or CONFIGR to change it on the system diskette.

ILLEGAL VALUE 53 35H

This message means that a command or program expected to receive a number within a certain range or having certain other properties (e.g., you might have a program that accepts only even numbers), and that the number you entered did not meet those requirements. Determine what values the program will accept, and then enter another value. (Depending on what kind of error handling is in force, you might have to execute the program again.)

MEMORY PROTECT 18 12H

This message is displayed if you try to load a program or data into memory that is occupied by the resident portion of PTDOS. If you need more memory space below the system and can compromise buffer space, use the SET BU= or CONFIGR command to raise the lowest system-managed buffer address. Otherwise, you must load the program or data elsewhere in memory.

NAME TOO LONG 16 1ØH

This message means that you specified a file name consisting of more than eight characters. Shorten the file name and enter the command again.

NO FILE ID'S 28 1CH

DID YOU REALLY GET THIS MESSAGE? Every time a file is created, it is assigned a unique file ID that is permanently attached to the file. (The file ID is not to be confused with a file NUMBER, which lasts only from the time a file is opened to the time that file is closed). There are 64K possible file ID'S; in order to see the NO FILE ID'S message, you have to have created more than 65,536 files! (Actually, they need not have been created on the same diskette, because whenever you make a copy of a diskette with DISKCOPY, the copy adopts the number of file ID's that existed on the source diskette, even if most of those files no longer exist.) If the diskette is very old, use GET to transfer files onto other diskettes. (You can continue to read from the old diskette, and even to change the existing files. The only restriction imposed by the NO FILE ID'S condition is that you can no longer create new files on the diskette.) If you see the message and have not created 65,536 files, suspect a hardware problem; follow the suggestions under "DEALING WITH HARDWARE-RELATED ERRORS," above.

NON-RESPONDING DRIVER 23 17H

This message can indicate that a device is not plugged in or connected properly. Check the condition of the device you are trying to use.

NO SPACE FOR BUFFER 27 1BH

This message means that there is not enough room in the system-managed buffer area to accommodate one block of the file you have just tried to open. If you were trying to open the file with a command from the keyboard, you should close a file you are not using, or close all open files by pressing the CTRL and C keys simultaneously. If the file was being opened by a system call in your own program, and if that program seems to require more buffer space than the present configuration of PTDOS will allow, use the SET BU= or CONFIGR command to lower GLLOW (the parameter that defines the lowest system-managed buffer address). The SET command will change the system in memory, but not on the diskette; the CONFIGR command will change the system on the diskette, but not in memory (until the next time you bootload PTDOS from that diskette).

READ-BACK FAILED 39 27H

This error can only occur if PTDOS is configured to verify all data that it writes to the diskette. (If a block can be read immediately after it is written, it is assumed to be verified.) The error is indicative of a hardware problem; follow the suggestions under "DEALING WITH HARDWARE-RELATED ERRORS," above. To configure PTDOS so that all data written are verified by a read-back check, use the SET SW=V command or the CONFIGR command to change the GLRBC flag in the System Global Area.

SECTOR CONFLICT 35 23H

This message means that hardware and software do not agree about what sector has been found on the diskette. Follow the suggestions under "DEALING WITH HARDWARE-RELATED ERRORS," above.

SYSTEM WRITE-LOCKED 40 28H

This message is displayed if the system is write-locked and you try to perform an operation that would alter data on a diskette in any disk drive unit. (If you have set the WRITE LOCK switch with SET SW=L or CONFIGR, you can not kill, endfile, create, write, or otherwise alter any file on any diskette.) To remove WRITE LOCK from the configuration of PTDOS on a system diskette, reset the computer and bootload from a system diskette that does not have a write-lock; then insert the write-locked diskette in unit 1, and use CONFIGR /1.

TOO MANY FILES OPEN 26 1AH

This message indicates that the number of files you have tried to open exceeds the maximum number of open files that can be supported in the current configuration of PTDOS. The CONFIGR command may be used to set the maximum number of open files to a number between 7 and 255. If you tried to open more files than the current configuration of PTDOS will allow, change the configuration on your diskette or determine why the files you open are not being closed properly. If a file is multiply open, it is regarded as more than one open file; for example, opening the same file three times (without closing it in the meantime) is exactly like opening three different files.

UNIT CONFLICT 19 13H

This message is returned when you have made contradictory references to disk drive units. For example, the command *RENAME POOH,PIGLET/1 is contradictory, because it is impossible to rename a file on one diskette so that it is suddenly on another diskette. (By contrast, it is possible to enter the command *RENAME PIGLET/1,POOH, because the command will assume that the new file name POOH will be given to the file PIGLET on unit 1.

USER MEMORY PROTECT 17 11H

This message means that you have attempted to load a program or data into an area of memory below the "lowest unprotected location" that you determined with SET PR= or CONFIGR. Change that address in memory (with SET) or on the diskette (with CONFIGR).

APPENDIX 1

GETTING STARTED WITH PTDOS

INTRODUCTION

This section is designed to allow a new user to get the feel of operating the HELIOS II system using PTDOS. All of the actions and instructions necessary are included to allow the user to perform some introductory operations with the system. These operations will help you to understand the other sections of the manual.

It is assumed that the HELIOS II is fully operational and has been tested as explained in the construction section of the manual. It is also necessary for the user to have at least 16K of memory configured as follows:

4K : 0000H - 0FFFH
12K : 9000H - BFFFH

16K of memory is just enough for use by PTDOS. Additional user memory is required for most applications.

Equipment needed:

A. Sol or other computer with 16K or more of memory
B. HELIOS II with interface cards
C. Keyboard input
D. Some form of output device (video preferred)
E. PTDOS system diskette

BRINGING UP PTDOS

Make sure all cables are connected, the computer is on and the power switch on the front of the HELIOS II cabinet is on. Also make sure that you have memory at the address locations indicated above, and that this memory is unprotected.

If you have not tested the HELIOS II as directed in the manual, do it now. The PTDOS system diskette will be used from here on, and it is possible to destroy the data on the diskette if the components are not functioning correctly.

INSERTING THE PTDOS SYSTEM DISKETTE

The PTDOS system diskette contains all of the software necessary for system operations. This includes the system resident code, the error utility files, and all of the command files. The diskette is labeled PTDOS 1.4 SYSTEM DISK.

Remove the diskette from its protective jacket, being careful not to touch the exposed areas of the actual recording surface, visible at

the center and through the drive-head access slots. See Figure 4-3, Diskette Orientation for Loading, in Section 4, Operating Instructions, of the harware manual, for positioning of the diskette prior to insertion. Now, insert the diskette into the left drive slot (unit Ø), and push it all the way into the slot until you hear the drive engage and lock the diskette into place. The diskette may be removed by pushing the left drive eject button at the lower left corner of the cabinet. Always remove diskettes from the drive slots before turning the power off or on.

THE BOOTSTRAP PROGRAM

The PTDOS (Processor Technology Disk Operating System) code is contained on the PTDOS system diskette, and the resident portion (the part that resides in the computer memory) must initially be read from the system diskette into the computer's memory. This is accomplished through the use of a BOOTSTRAP program. This program is contained in ROM on the BOOTLOAD Personality Module, Order No. 1Ø7Ø15.

A listing of this short program is given at the end of this appendix along with a dump of code for use at address 4ØØ hex. Bootstrap can be placed in PROM or loaded from cassette tape, if you do not have the special Personality Module.

If you have a Sol computer with the BOOTLOAD Personality Module, type the command "BO" in Solos command mode. Otherwise, load the bootstap object code into the computer memory starting at the proper location, using your computer's monitor (such as SOLOS on the Sol) to enter the bootstrap object code, or use whatever method your computer provides. It is possible to save the loader at this point using the SOLOS or CUTER tape routines. Loading the bootstrap program from a tape will save time and effort in bringing the system up after it has been turned off.

Check this bootstrap code you have just entered and make sure it is correct. It is possible to damage the programs on the diskette with improper code in the loader.

When you are satisfied with the bootstrap code, start the computer running at location 8ØØ Hex. If you used the "BO" command with the BOOTLOAD Personality Module, the program will be run automatically.

The system resident code will now be loaded into locations 9ØØØH - BFFFH, and control will be passed to this code. The system diskette name, revision date, and other information will be printed, followed by the PTDOS prompt character (*).

The prompt indicates that PTDOS is now waiting for input.

If you do not see the above information printed, remove the diskette by pushing the eject button at the lower left of the HELIOS II cabinet, and go back and check all of the information presented so far. Make sure you understand the procedures, and make sure the HELIOS II has passed the disk test routines. The system diskette will not work if it has been used for a disk test, written on in an

incorrect manner or damaged in some other way. Try the bootstrap procedure again.

CARRIAGE RETURN <cr>:

The characters typed into PTDOS are stored in a buffer until a carriage return is typed. In the examples below, a carriage return is indicated by the symbol: <cr>

The input line of characters is not acted upon until this carriage return is typed.

BACKSPACE

If you make a mistake in typing, you may backspace to the character that is incorrect by typing the 'DEL' key (7FH).

NOTE: The delete key is NOT the same as shift underline.

The entire line may be deleted by typing control-X, i.e, by holding down the 'CTRL' (control) key and typing the 'X' key while it is held down. PTDOS will respond by echoing an exclamation point (!) at the end of the line just deleted. At this point the line may be entered again.

DISKCOPY

As mentioned elsewhere in the manual, it is important to make a backup copy of this system diskette. It would be a good idea to do this now. If you wish to make a copy now, follow the instructions below, otherwise, skip to the next section - "FILES EVERYWHERE."

Insert a blank diskette into the right hand drive (unit 1) in the same manner as the system diskette was put into unit Ø. The diskette label is to the left-rear-top with your right-hand thumb about over the label.

Now - the prompt character has been printed, and PTDOS is waiting for command input. Type the following command:

```
DISKCOPY/1,S=C <cr>
```

This command will "condition" the new diskette so that the Helios II Disk Memory System can use it. The command will ask you to type a carriage return (C/R) to continue. If you typed the command correctly, press the return key; otherwise, press the MODE key (or the CTRL and @ keys) and try again.

```
DISKCOPY Ø,1 <cr>
```

This command tells PTDOS to make a copy of the diskette in unit Ø onto the diskette in unit 1. If you have made a mistake in typing the command, an error message will be printed, followed by a prompt (*). Type the command again.

The system will now print the following:

```
DISKCOPY UNIT Ø TO 1
C/R TO CONTINUE, ESC TO ABORT:
```

Check to make sure the command said to copy unit Ø to unit 1. If this is correct, type a carriage return. The diskcopy program will now be loaded into memory, and a copy of the system diskette will be made onto the diskette in unit 1. This takes about 5-6 minutes. If any errors are made during the diskcopy, a message will be printed. Usually the program will retry the section that gave the error, and continue on.

When the copy is complete, a message to that effect will appear, and the prompt character will be typed. Remove the diskette in unit Ø (left hand) and store it in a safe place. The diskette in unit 1 should now be placed in unit Ø for the remaining operations. You have just made a "backup" System diskette-a precaution that can save the results of much labor in the event of an inadvertant command, a rampant program, power failure, equipment malfunction, or diskette damage.

Please note that PTDOS and most software available from Processor Technology and other manufacturers is subject to legal protection including copyright. Unauthorized circulation of copies, or possession of "bootleg" copies may constitute a crime.

FILES EVERYWHERE

It is important to understand the concept of files. To PTDOS, almost everything is a file. The keyboard is considered to be an input file, the display screen (or printer) is an output file, commands are files, anything saved on the disk is a file, and so on. All major communication to and from PTDOS is done through files.

We are now going to try a few operations. If the message DRIVE NOT READY appears at any time during this section, make sure the system diskette is inserted correctly in unit Ø. Also, go back and check to make sure everything is in working order, according to other sections in the manual.

Also, if an error message is printed, look at what you had typed as a command - something may be wrong with it.

We will now go through a few examples in which you will type commands to PTDOS. The characters you should type in from the keyboard will follow the word 'TYPE:' in all of the examples. In the first example below - TYPE: CREATE ... you would start typing with the word CREATE. Also remember that you may backspace to correct a character by typing the 'DEL' key, or the entire line may be deleted by typing control-X. If an error was made in the line typed, PTDOS will respond with an error message, then the prompt (*). Just type the line in again. Remember to follow all commands with the RETURN key <cr>.

Now, let's look at creating a new file.

```
TYPE:  CREATE TEST <cr>
```

In the standard operating mode, PTDOS has printed the prompt (*), and is reading file #Ø (the keyboard). This is the default input file to the CI (command interpreter). Characters typed into file #Ø are echoed back out to file #1 (display device), the default output file. (Note: files #Ø and #1 are always open and cannot be closed.) The characters you type in on the keyboard are stored into a line buffer until a carriage return <cr> is typed. The line is then decoded by the CI (command interpreter). Remember, the CI is just reading a file, and right now it happens to be the keyboard. Now the CI expects commands, so starting at the front of the line just typed in, leading blanks are skipped over, and then characters are saved until the first 'space' character (blank) is found.

This first group of characters is assumed to be a command name.

Finding you have typed 'CREATE,' the disk directory is searched for a file called CREATE. Notice that all command names are just file names! Upon finding that a file called CREATE does indeed exist, it is now loaded into the computer's memory. The load address (and run address if any) is included as part of the file's data. The file, CREATE, loads into an area of the PTDOS system called the CXBUF. This is the 'command execution buffer' - many of the commands load into this buffer. CREATE also includes a run address, so after it has been loaded into the CXBUF, control is transfered to the CREATE code. ('Control is transferred' means that the 8Ø8Ø's program counter has been addressed by one of the program counter altering 8Ø8Ø instructions, such as JMP, CALL, PCHL, etc.) The CREATE code now has control; this code expects parameters in a certain order to perform its function of creating a new file, and it gets these parameters from the current input file. The current input file number is retrieved from PTDOS (in this case, the CI file), and the necessary parameters are read. The CI file has a pointer which is now pointing one past the first space in the last command line typed in. (CREATE TEST)

The first parameter required by the CREATE command is the name of the file to be created. In the example, there are no other parameters present, so the default values for file type (.) and block size (4CØ Hex) are used. The CREATE code now calls into the PTDOS entry area (SYS) with the necessary values for the CREATE function. PTDOS now has control, and the CREATE function is evoked (parameters checked, etc.), and a new file called TEST is created. The following is a brief explanation of the actions that this procedure entails.

The FSMAP (free space map - file) is read into memory and checked for the first free block of 4CØH size. The disk address (sector and track) is noted, and the FSMAP is updated to show this area is currently in use. The FSMAP is then written back onto the diskette. Now the diskette file directory (DIRECTRY) is read into memory and the file named TEST along with the track, sector, file type, block size, is added. The directory is then written back onto the diskette. Other associated system maintenance is taken care of, and control is passed back to the CREATE code. Errors (if any) will be reported now, and control will be passed to the PTDOS system, which will continue to read from the CI file if there are multiple commands. If there are no more commands in the CI, a prompt will be printed, and PTDOS will wait for more input.

Some commands report back on some aspects of their operation. This information is written out to the current output file (in this case, the default file #1, the CO - command output file). An An example of this output will be seen in the next command.

```
TYPE:  KILL TEST
```

The file will be looked up, checked to make sure it is not kill-protected, and the space on the diskette used by the file TEST will be released, and the file name removed from the directory. The contents of the file are not actually erased, but since the space containing the file is defined as free, subsequent write operations can write over the space.

Input may come from any file and not just the keyboard (file #Ø). Next, we will CREATE a file, write some commands into it, and then use this new file as the input file.

```
TYPE:  CREATE PROC,P,1Ø0; CREATE DIRC,P,1Ø24:D  <cr>
```

NOTE: At least one space (blank) is necessary following a command name. All other spaces are ignored.

Notice that we have used multiple commands in the line just typed. The commands are separated by a semicolon (;). Also, in the second file, the block size was typed in decimal, as indicated by the (:D) following the block size number. (Hexadecimal is the default number base).

The file PROC has now been created. Now, we will write directly onto this file. We will do this by copying from file #Ø, the keyboard, to the file, PROC, just created.

A PROCEDURE FILE EXAMPLE:

```
TYPE:  COPY #Ø,PROC <cr>

TYPE:  SET SW=-E <cr>
TYPE:  $PR;$PR  LISTING  FILES T=P  TO THE FILE DIRC;$PR <cr>
TYPE:  SETOUT DIRC;FILES T=P <cr>
TYPE:  SETOUT #1 <cr>
TYPE:  $PR; $PR  TYPE CARRIAGE RETURN TO CONTINUE <cr>
TYPE:  SET SW=E;$WAIT <cr>
TYPE:  FILES S=-I <cr>
TYPE:  OPEN? <cr>
TYPE:  FREE? <cr>
TYPE:  SETIN #Ø <cr>
```

Lastly, type control-C. This will reset the system and close the input to the file PROC. Also notice that multiple commands on one line are possible here too, just as with any other input line.

To check on what we have written into the file PROC, use the following:

```
TYPE: COPY PROC,#1
```

This will copy the file PROC to the current output file. Look at the list of commands in the file PROC, and check to make sure they are correct. If a mistake was made, go back to 'A PROCEDURE FILE EXAMPLE' and start over.

Now, we have created a file consisting of other commands. This may be thought of as a procedure file. Since the CI does not know (or care) where the commands it reads come from, we shall tell it to read from the file PROC. The commands that are contained will be executed, just as if they had been typed directly in from the keyboard!

Here are the actions that will take place after we switch the input to PTDOS:

First, the command SET SW=-E will turn off the command echo. Next, the output file will be changed to the file DIRC, and the FILES command will list all of the files of the type 'P' out to that file. The output file will then be switched back to the consol print device (file #1). The $PR command will now print its message, and the echo mode will be turned on by the SET SW=E command (the commands read by the CI will be echoed to the output file). Now the $WAIT command will be echoed, and the system will wait for a carriage return to be typed in from the keyboard. To continue, type the carriage return. FILES S=-I will now be executed. The S=-I tells the files command to list all files on the diskette, including those files with the information protect attribute (I). Next, the OPEN? command will list all files currently open. (Files #Ø,#1,#2 are not listed - they are always open and cannot be closed.) The FREE? command returns the number of unused sectors on the diskette. The final command read from our procedure file switches the input file from itself back to the keyboard. The PTDOS prompt is printed, and the system is now once again waiting for more input.

Now, let's switch the input file from the keyboard to our new file, PROC. PTDOS will now read the command lines from PROC, and execute the commands in order. The input file will be switched back to the keyboard after the SETIN #Ø command has be executed. If a mistake has been made in the format or typing of the command lines in PROC, an error message will be printed, and the input will be automatically switched back to the default input file - the keyboard. An error of this type may leave files open - to make sure all files are closed, type control-C. Then go back to the text of the procedure file above, starting with the COPY #Ø,PROC command, and type all of the lines again. Be careful to type exactly what is shown - the spelling and punctuation must be correct! Don't forget the last control-C.

TYPE: SETIN PROC <cr>

In the above example, a FILES T=P command was executed with the list written to the file DIRC. To look at the information now contained in the DIRC file:

TYPE: COPY DIRC,#1 <cr>

(When a file is opened, a file number is associated with the buffer for the file. To access an open file, a number sign (#) followed by the file number is used - such as file #1 above. If a file is not open, the file name is used, as with the file DIRC in the above example.)

The procedure file PROC that we have written may be used as the input file as many times as you wish. Just switch the input to PTDOS as above. If you have no further need of the files DIRC and PROC, they may be killed at this time.

TYPE: KILL DIRC,PROFF,PROC <cr>

(Note the effect of trying to kill the non-existent file. PROFF.)

If the kill command responds with <IS OPEN>, the file to be killed is still open. A file must be closed to be deleted. Typing Control-C closes all open files. Once a file is killed it is gone forever, so do not kill files you still need.

In the above example, the same purpose could have been served by using the "DO" Command Interpreter Macro Facility, as described in Chapter 7. "DO" provides an even more flexible way of creating procedure files.

NOTES

The above demonstration was designed to help the user to understand a little of the 'files' concept, and at the same time let the user perform some actual PTDOS operations. Read the rest of the manual for complete information on both the system supplied commands and information on all of the PTDOS - HELIOS II operations.

Backup Your Diskettes

Keep your disks orderly - use a daily work disk, and when finished working with a file or files, copy them to another diskette for storage.

CLOSE /	will close all open files
The 'DEL' key	will backspace the input cursor
Control / X	will delete an entire input line
Control / C	if typed as the first character of a line will close all files and execute a system reset. A file may may be opened more than once. If a file is open, it cannot be end-filed or killed. If you get errors during operations, and you are not sure of the current status of open files, type control /X followed by a control C. All files will be closed, and the input file will be empty.

It is possible for a user program to write into the PTDOS system resident area by mistake. If you think this may have happened, type BOOTLOAD <cr>. The system will then be reloaded.

APPENDIX 2

BOOTSTRAPPING

The system command BOOTLOAD reinitializes PTDOS by loading the resident code into memory from the system diskette. After a power interruption or major system crash, it is necessary to run a short bootstrap program that in turn loads and runs the BOOTLOAD program.

On a Sol Terminal Computer with a BOOTLOAD personality module, the bootstrap program is built into the monitor and may be invoked by the BOOT command.

On a Sol with a SOLOS personality module, or on another computer using the CUTER monitor program, the SOLOS/CUTER command ENTR 800 may be used to enter the program by hand from the listing that follows. To save the BOOTLOAD program on a cassette, type

SAVE BOOT 800 84E<cr>

immediately after entering the program. Thereafter, BOOT may be loaded and executed from SOLOS/CUTER with the command

XEQ BOOT<cr>

If you wish to burn the bootstrap program into PROM, you may give it any origin at or above 800H. Normally, an address above 0BFFFH (the highest address assigned to PTDOS) will be chosen.

BOOTSTRAP PROGRAM LISTING

```
0800                            ORG     800H
0800 3E CF          BOOTS       MVI     A,0CFH
0802 D3 F7                      OUT     0F7H
0804 D3 F5                      OUT     0F5H
0806 3E FF                      MVI     A,-1
0808 D3 F1                      OUT     0F1H
080A DB F0          BOOTL       IN      0F0H
080C E6 40                      ANI     40H
080E C2 0A 08                   JNZ     BOOTL
0811 3E DF                      MVI     A,0DFH
0813 D3 F7                      OUT     0F7H
0815 DB F0          IFIN        IN      0F0H
0817 07                         RLC
0818 DA 15 08                   JC      IFIN
081B 11 18 03                   LXI     D,1290H/6
081E 1B             IFIN2       DCX     D
081F 7A                         MOV     A,D
0820 B3                         ORA     E
0821 C2 1E 08                   JNZ     IFIN2
0824 DB F0          IFIN3       IN      0F0H
0826 07                         RLC
0827 DA 24 08                   JC      IFIN3
082A DB F0          SWAIT       IN      0F0H
082C E6 02                      ANI     2
082E CA 2A 08                   JZ      SWAIT
0831 3E 40                      MVI     A,40H
0833 D3 F3                      OUT     0F3H
0835 3E 03                      MVI     A,3
0837 D3 F4                      OUT     0F4H
0839 AF                         XRA     A
083A D3 F5                      OUT     0F5H
083C D3 F6                      OUT     0F6H
083E 3E 03                      MVI     A,3
0840 D3 F1                      OUT     0F1H
0842 DB F0          DLOOP       IN      0F0H
0844 E6 0B                      ANI     0BH
0846 CA 42 08                   JZ      DLOOP
0849 E6 08                      ANI     8
084B C2 00 08                   JNZ     BOOTS
084E C3 04 00                   JMP     4
```

APPENDIX 3

COMMAND SUMMARY