UCSD p-System™ Operating System Reference Manual

Texas Instruments Professional Computer

Cover Part No. 2232343-0001

UCSD p-System™ Operating System Reference Manual
TI Part No. 2232395-0001
Original Issue: 15 April 1983

Preface

This publication is a reference manual for the UCSD p-System™* on the Texas Instruments Professional Computer. It covers the operating system, filer, Screen-Oriented Editor, and several utilities. It describes the facilities of the major p-System components and provides basic instructions for using them. If you are slightly familiar with the p-System, the information presented here will complement and increase your knowledge of it. However, if you are a beginner or have never used this system, you should first read:

Personal Computing with UCSD p-System
(Part Number 2232418-0001)

For further information about the system and its use, refer to the following publications:

UCSD p-System Introduction
(Part Number 2232396-0001)

UCSD p-System Program Development
(Part Number 2232399-0001)

UCSD p-System Assembler
(Part Number 2232402-0001)

UCSD p-System Internal Architecture
(Part Number 2232400-0001)

*UCSD Pascal™**
(Part Number 2232401-0001)

* UCSD p-System and UCSD Pascal are trademarks of the Regents of the University of California.

DISCLAIMER

This document and the software it describes are subject to change without notice. No warranty expressed or implied covers their use. Neither the manufacturer nor the seller is responsible or liable for any consequences of their use.

Contents

1

Introduction

HOW TO USE THIS MANUAL

Each part of this manual is designed to help you get the most out of the p-System on the Texas Instruments Professional Computer.

The preface describes the audience and the purpose for which the manual was written. It also lists other books that may be of interest to you.

Chapter 1, Introduction, presents background information about the p-System, including a short history of p-System development and a description of p-System components.

Chapter 2, The Operating System, explains the menus, prompts, and commands used by the p-System's operating system. It also shows how to swap disk volumes while a program is running.

Chapter 3, File Management, presents considerable information about file organization and file handling, as well as descriptions of the filer commands.

Chapter 4, System Editor, describes the Screen-Oriented Editor.

Chapter 5, Utility Programs, describes several programs which provide support to the Texas Instruments Professional Computer.

The appendices present useful reference material:

A Bootstrapping the UCSD p-System
B Special Keys
C Execution Errors
D I/O Results
E Device Number Assignments
F ASCII Code
G Extended Memory on the Texas Instruments Professional Computer
H Release Disk Configurations

BACKGROUND

In June 1979, SofTech Microsystems in San Diego began to license, support, maintain, and develop the p-System. The resulting effort to build the world's best small computer environment for executing and developing applications has dramatically increased the growth and use of the p-System. The first p-System ran on a 16-bit microprocessor. Today, the p-System runs on 8-bit, 16-bit, and 32-bit machines—including the Z80™[1], 8080/8085, 8086, 6502, 6809, 68000, 9900, PDP-11™[2], LSI-11™[2], and VAX™[2].

The p-System began as the solution to a problem. The University of California at San Diego needed interactive access to a high-level language for a computer science course. In late 1974, Kenneth L. Bowles began directing the development of the solution to that problem: the p-System. He played a principal role in the early development of the software.

In the summer of 1977, a few off-campus users began running a version of the p-System on a PDP-11. When a version for the 8080 and the Z80 began operating in early 1978, outside interest increased until a description of the p-System in *Byte* magazine drew over a thousand inquiries.

As interest grew, the demand for the p-System could not be met within the available resources of the project. SofTech Microsystems was chosen to support and develop the p-System because of its reputation for quality, high technology, and language design and implementations.

Now the p-System is available on the Texas Instruments Professional Computer.

[1] Z80 is a trademark of Zilog, Incorporated.

[2] PDP-11, LSI-11, and VAX are trademarks of Digital Equipment Corporation.

DESIGN PHILOSOPHY

The development team members, many of whom continued their efforts on behalf of the system at SofTech Microsystems, decided to use stand-alone, personal computers as the hardware foundation for the p-System rather than large, time-sharing computers. They chose Pascal for the programming language because it could serve in two capacities: the language for the course and the system software implementation language.

The development team had three primary design concerns:

- The user interface must be oriented specifically to the novice, but must be acceptable to the expert.

- The implementation must fit into personal, stand-alone machines (64K bytes of memory, standard floppy disks, and a CRT terminal).

- The implementation must provide a portable software environment where code files (including the operating system) could be moved intact to a new microcomputer. In this way, application programs written for one microcomputer could run on another microcomputer without recompilation.

The current design philosophy at SofTech Microsystems, where the p-System continues to evolve, is basically the same as the original philosophy.

User-Friendly

The p-System continuously identifies its current mode and the options available to you in that mode. This is accomplished by using menus, displays, and prompts. You may select an option from a menu by pressing a single-character command. The system's displays then guide your interactions with the computer. As you gain more experience, you can ignore the continuous status information—unless it is needed.

Portable

The p-System is more portable than any other microcomputer system. It protects your software investments without restricting hardware options. The p-System does this by compiling programs into p-code —rather than native machine language—thus allowing these code files to be executed on any microcomputer that runs the UCSD p-System.

USING THE p-SYSTEM

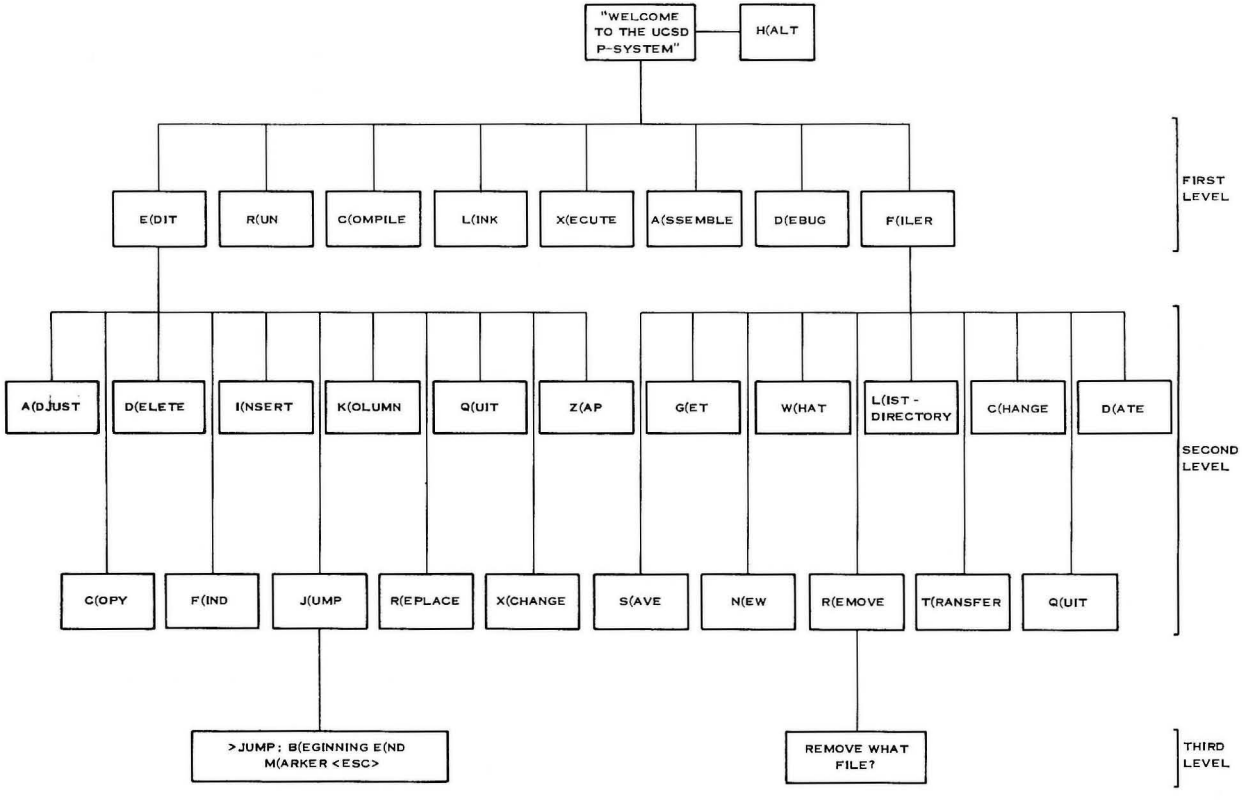
The p-System includes an operating system, filer, editor, and several other components. The filer, editor, and other components are separate programs that perform functions traditionally performed by an operating system.

Menus and Prompts

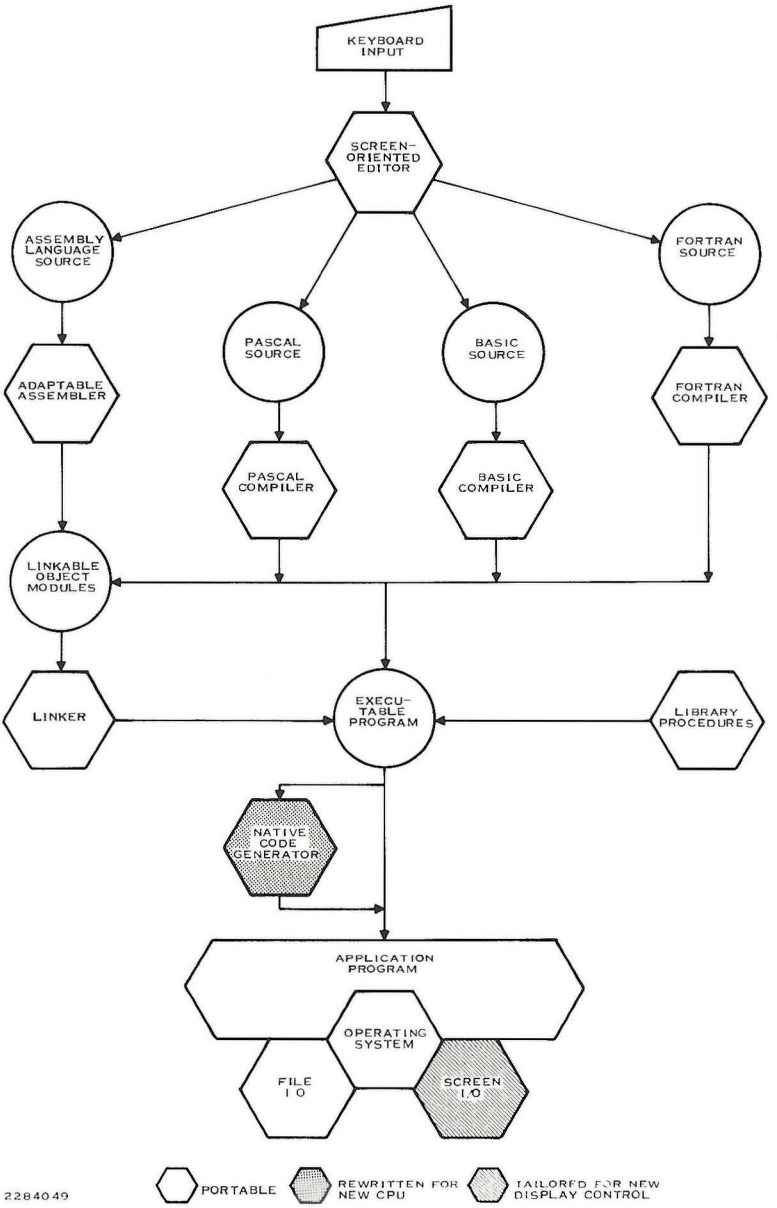
The p-System is menu-driven; that is, it displays a menu at the top of the display unit that lists the available commands. To use any one of these commands, you need press only one key. Often, prompts are displayed. They require you to enter in a response and then press the RETURN key. You can use the BACKSPACE key if you make a mistake while responding to a prompt.

The menus and prompts are organized in a hierarchy (see the figure on page 1-7). The outermost (command) menu lists several items, including E(dit. When you press the E key to call the E(dit option, the p-System activates the editor. To quit using the editor, press the Q key for Q(uit; this will return you to the command menu.

The figure on page 1-8 graphically describes the interrelationships of the major p-System components.


"WELCOME TO THE UCSD P-SYSTEM"
H(ALT
E(DIT
R(UN
C(OMPILE
L(INK
X(ECUTE
A(SSEMBLE
D(EBUG
F(ILER
FIRST LEVEL
A(DJUST
D(ELETE
I(NSERT
K(OLUMN
Q(UIT
Z(AP
G(ET
W(HAT
L(IST - DIRECTORY
C(HANGE
D(ATE
SECOND LEVEL
C(OPY
F(IND
J(UMP
R(EPLACE
X(CHANGE
S(AVE
N(EW
R(EMOVE
T(RANSFER
Q(UIT
>JUMP: B(EGINNING E(ND M(ARKER <ESC>
REMOVE WHAT FILE?
THIRD LEVEL

2284048


KEYBOARD INPUT
SCREEN-ORIENTED EDITOR
ASSEMBLY LANGUAGE SOURCE
PASCAL SOURCE
BASIC SOURCE
FORTRAN SOURCE
ADAPTABLE ASSEMBLER
PASCAL COMPILER
BASIC COMPILER
FORTRAN COMPILER
LINKABLE OBJECT MODULES
LINKER
EXECU-TABLE PROGRAM
LIBRARY PROCEDURES
NATIVE CODE GENERATOR
APPLICATION PROGRAM
OPERATING SYSTEM
FILE I O
SCREEN I O
2284049
PORTABLE
REWRITTEN FOR NEW CPU
TAILORED FOR NEW DISPLAY CONTROL

System Files

The system files are disk files which contain the bulk of the UCSD p-system. Most of the system files reside on the system disk, which is the disk you bootstrap with.

These files are listed as follows:

SYSTEM.PASCAL
SYSTEM.INTERP
SYSTEM.MISCINFO
SYSTEM.LIBRARY
SYSTEM.MENU
SYSTEM.STARTUP
SYSTEM.SYNTAX

The following system files do not necessarily need to reside on the system disk:

SYSTEM.COMPILER
SYSTEM.ASSMBLER (no E)
SYSTEM.EDITOR
SYSTEM.FILER

SYSTEM.PASCAL is the operating system.

SYSTEM.MISCINFO is a data file that contains miscellaneous information about an individual system. This includes terminal handling, memory configurations, and miscellaneous options.

SYSTEM.EDITOR contains the editor that you can call by pressing E for E(ditor—as displayed on the command menu.

SYSTEM.COMPILER contains the Pascal compiler.

SYSTEM.ASSMBLER is the assembler that translates assembly language into 8086 machine code. The assembler needs these three files: 8086.OPCODES, 8086.ERRORS, and 8087.FOPS.

SYSTEM.SYNTAX contains the Pascal compiler's error messages. It must be on the boot disk if you want to have compile-time errors displayed in English rather than as error numbers.

SYSTEM.LIBRARY contains previously compiled or assembled routines that can be used by other programs. Long integer support routines are usually found here.

SYSTEM.STARTUP is an executable code file. If a file called SYSTEM.STARTUP is present when the system is bootstrapped or initialized, the p-System executes it before the command menu is displayed.

SYSTEM.MENU, like SYSTEM.STARTUP, can be any executable code file. If it is present on the boot disk, it is executed every time the command menu is about to be displayed. This is generally used for turnkey applications.

If both files are present, SYSTEM.STARTUP executes before SYSTEM.MENU. Since SYSTEM.STARTUP is executed whenever the system is initialized or bootstrapped, it might be used to perform periodic initializations, such as setting the date and time, or it could be an application program that executes without any user interaction with the operating system. The p-System executes SYSTEM.MENU instead of displaying the command prompt line. It is intended to allow for custom menus or prompts.

SYSTEM.INTERP is the assembly language program that emulates the p-machine on the host processor. The following are some other possible names for these emulators, which are usually machine-specific:

SYSTEM.PDP-11
SYSTEM.ALTOS
SYSTEM.HEATH

2

The Operating System

INTRODUCTION

This chapter describes the UCSD p-System's operating system on the Texas Instruments Professional Computer. The operating system is the core of the p-System. When you first boot the p-System, the operating system's menu appears. From here, you can select other major p-System components or run programs. Each time a p-System component or a program finishes execution, you are returned to the operating system's menu.

The operating system's menu is called the command menu. The items on it include the editor, filer, compiler, and more.

This chapter describes how menus and prompts are used by the p-System. It goes on to describe the particular items on the command menu.

MENUS AND PROMPTS

Menus

The following describes the menus used by the p-System.

- The first word (title) of the menu identifies the level of the menu, for example, command or edit.

- The selections available on a menu are located to the right of the menu title. The letter denoting the key that selects an option is capitalized and set off from the rest of the word with a parenthesis.

- The version number of the system is listed at the end of the line in square brackets.

- A question mark on the right of a menu indicates that there are more items on the menu than can fit on a single line. Entering ? causes more of the menu to be displayed.

Some typical menus are listed as follows:

```
Command: E(dit, R(un, F(ile, C(omp, L(ink, X(ecute,
  A(ssem, D(ebug,  ? [version]
Filer: G(et, S(ave, W(hat, N(ew, L(dir, R(em, C(hng,
  T(rans, D(ate, ? [version]
>Edit: A(djust C(opy D(el F(ind I(nsert J(ump K(ol
  M(argin P(age ? [version]
```

If you press the ? key for the command menu, the following is displayed:

```
Command: H(alt, I(nitialize, U(ser restart, M(onitor
  [version]
```

Selecting an option at the command menu produces one of the following results.

- The p-System allows you to execute a program.

- A p-System component is started; for example, the filer or editor.

- The system alters its state; for example, as when you select H(alt.

In general, you may exit from the system by pressing the Q key to call Q(uit. After performing a function, you may press the space bar to clear the screen and redisplay the menu.

Prompts

As just discussed, a menu displays options you can select with a single keystroke; however, a prompt requests information from you. For example, if you want to execute a program, you would select the X(ecute option from the command menu by pressing the X key. The system will respond with the following request—called a prompt:

```
Execute what file?
```

Your response to this would be to enter the name of the program to be executed and then to press the RETURN key.

If you make an error while entering your response, you can press the BACKSPACE key to correct it. You can then resume entering the correct response.

Another example of a prompt is shown when the filer is used to list the directory of a volume. After you press the F key as listed on the command menu to display the filer menu, and then press the L key as listed on the filer menu, the following prompt will be displayed:

```
Dir listing of what vol?
```

Your response to this prompt would be to enter any valid volume name and then to press the RETURN key.

Often prompts require that you enter a file name. File names (as described in Chapter 3) often end with specific suffixes such as .TEXT or .CODE. Usually, in response to a prompt, you should omit these suffixes. The system programs append them automatically. To prevent automatic appending, place a period at the end of the file name.

When a program—such as a compiler—requires both a source text file and a destination code file name, the code file name may be given as $. This indicates the same name as the text file with .CODE appended instead of .TEXT. Alternatively, you can use $., which is the source file name exactly.

For example, press the A key to select the A(ssemble command. The system then displays the following prompt.

```
Assemble what text?
```

Enter YOUR.FILE and press the **RETURN** key. If your .FILE.TEXT exists, the system will display the following prompt.

```
To what code file?
```

Enter $ and press the **RETURN** key.

The preceding sequence assembles the file YOUR.FILE.TEXT and places the resulting code in YOUR.FILE.CODE.

You may also use device names when responding to certain prompts. For example, the assembler next displays this prompt:

```
Output file for assembled listing: (<CR>) for none)
```

You could enter PRINTER: and press the **RETURN** key. The printer is a device (not a file). The assembled listing is sent there.

DISK SWAPPING

Since the operating system swaps code segments into and out of main memory while a program is running, and since you may change disks at various times, the operating system has various checks to aid you in handling disks, thus reducing errors.

When a program requires a code segment from a disk, but the disk containing the code segment is no longer in the drive, the operating system displays the following error message on the bottom of the display unit:

```
Need Segment SEGNAME: Put volume VOLNAME in unit U
  then type <space>
```

In the preceding example, the system could not find the disk VOLNAME and waited for you to press the space bar. (If you press the space bar but have not replaced VOLNAME, the system will re-display the error message.)

OPERATING SYSTEM COMMANDS

This section covers the items on the command menu in alphabetical order. Most of these items are described in greater detail elsewhere.

In particular, the filer is described in Chapter 3 of this manual, and the editor is covered in Chapter 4.

The assembler and linker are covered in *UCSD p-System Assembler.*

The compiler and debugger are covered in *UCSD p-System Program Development.*

A(ssemble

On the menu: A(ssem

The A(ssemble command starts the assembler SYSTEM.ASSMBLER (notice that there is a missing E). If a work file is present, then the file *SYSTEM.WRK.TEXT or the designated .TEXT file is assembled to a code file of 8086 machine code. If there is no work file, the system displays a request for a source file, a code file, and a listing file; the defaults for these are *SYSTEM.WRK.TEXT, *SYSTEM.WRK.CODE, and no listing file.

If you simply press the RETURN key for the source file, the assembly is aborted. Similarly, if you press the ESC key and then press the RETURN key for the code file or listing file, the assembler is exited.

If the assembler encounters a syntax error, it displays the error number and the source line in question. It also displays an error message (if the file *8086.ERRORS is present). It gives you some options:

```
Error ##: error message
  <sp>(continue), <esc>(terminate), E(dit
```

You may continue the assembly by pressing the space bar; abort the assembly by pressing the ESC key or, by pressing E, return directly to the editor to correct the source file.

The assembler is described in *UCSD p-System Assembler.*

C(ompile

On the menu: C(omp

The C(ompile command starts the compiler, SYSTEM.COMPILER. If a work file is present, either *SYSTEM.WRK.TEXT or the designated text file is compiled to p-code. If there is no work file, the system displays a request for a source file and a code file; the defaults for these are *SYSTEM.WRK.TEXT and *SYSTEM.WRK.CODE. If you press the RETURN key for the code file, the default code file is *SYSTEM.WRK.CODE. If you press the RETURN key for the source file, the compilation is aborted. If you press the ESC key followed by the RETURN key for the code file, the compilation is aborted.

Next, the compiler asks for a listing file. This may be a disk file or communications volume. The default is *SYSTEM.LST.TEXT. If you press the ESC key followed by the RETURN key, the compilation is aborted.

If the compiler encounters a syntax error, it displays the error number, the source line in question, and the following menu.

```
Error ##
Line ##
Type <sp>(continue), <esc>(terminate), or 'E' to
  e(dit
```

You may continue compilation by pressing the space bar, abort compilation by pressing the ESC key, or proceed directly to the editor to correct the source file by pressing the E key. In the latter case, the editor will position the cursor where the error was detected.

If the file *SYSTEM.SYNTAX is present, the Pascal compiler displays a relevant error message instead of the error number.

The Pascal compiler is described in *UCSD p-System Program Development* and *UCSD Pascal.*

D(ebug

On the menu: D(ebug

This command starts the symbolic debugger. The debugger resides within SYSTEM.PASCAL. If your copy of SYSTEM.PASCAL does not contain the debugger, you need to use the Library utility (described in *UCSD p-System Program Development*) to place DEBUGGER.CODE into SYSTEM.PASCAL.

The symbolic debugger is a tool for debugging compiled programs. You can call it from the command menu or while a program is executing (when a breakpoint is encountered). Using the symbolic debugger, you may display and alter memory, single-step p-code, and do several other useful debugging operations.

To use the debugger effectively, you must be familiar with the UCSD p-machine architecture and must understand the p-code operators, stack usage, variable and parameter allocation, and so on. These topics are discussed in *UCSD p-System Internal Architecture.*

For more information about the symbolic debugger, refer to *UCSD p-System Program Development.*

E(dit

On the menu: E(dit

This command starts the editor, SYSTEM.EDITOR. If a .TEXT work file is present, the system indicates its availability for editing. If no work file is present, the system displays a request for a file name along with the option to escape from the editor, or to enter the editor with no file (with the intent of creating a new one).

Use the editor to create either program files or document text files and to alter or add to existing text files. (Refer to Chapter 4, System Editor, in this manual, for more information about the editor.)

F(ile

On the menu: F(ile

This command starts the filer, SYSTEM.FILER. The filer provides commands for managing files, manipulating work files, and maintaining disk directories. (Refer to Chapter 3, File Management, for detailed coverage of the filer.)

H(alt

On the menu: H(alt

This command reboots the system if there is a valid system disk on line.

It is not necessary to use H(alt when you are finished using the system. Just remove any diskettes and turn off the power.

NOTE

If you are using the optional RAM disk and want to save its files, copy those files to another disk before turning off the power.

I(nitialize

On the menu: I(nit

This command reinitializes the p-System.

*SYSTEM.STARTUP is executed, if present. SYSTEM.STARTUP must be a code file; it is executed automatically after a bootstrap or an I(nit command. If SYSTEM.MENU is present, it is then executed.

All run-time errors that are not fatal cause the system to initialize in the same manner as I(nitialize. At initialize time, much of the system's internal data is rebuilt, and SYSTEM.MISCINFO is reread.

An I(nitialize command does not clear any I/O redirection, but a run-time error reinitialization does.

L(ink

On the menu: L(ink

This command starts the linker, SYSTEM.LINKER. The linker allows you to link assembled machine code routines into host compilation units (compiled from a high-level language). It also allows you to link native code routines together. It is described in *UCSD p-System Assembler.*

M(onitor

On the menu: M(on

This command invokes the monitor. The monitor helps you to create *script files* which drive the system automatically. While in the monitor mode, you may use the p-System in a normal manner, but all your input is saved in the script file. Later, you can redirect the p-System's input to that file and your actions at the keyboard are reproduced.

Press the M key to start the M(onitor command. The system then displays the following menu.

```
Monitor: B(egin, E(nd, A(bort, S(uspend, R(esume
```

Press the B key to select the B(egin option. The system then requests a file name where it will store your sequence of commands. Enter the file name and press the RETURN key. Then R(esume and use whatever p-System commands you wish. When you are finished, select M(onitor again. Press the E key to call the E(nd option.

All your input is saved in the file you named. To use this file, redirect the system input to it with the I = execution option string.

B(egin starts a monitor. If a monitor file has already been opened, the system displays an error message.

E(nd terminates a monitor session and saves the monitor file. (You must use S(uspend or R(esume to return to the command menu.)

A(bort ends a monitor but does not save the monitor file. (You must use S(uspend or R(esume to return to the command menu.)

S(uspend turns off monitoring but does not close the monitor file. In other words, you are returned to the command menu where you can now enter commands without recording them. The monitor file remains open and in a state in which you can add to it by using R(esume.

R(esume starts monitoring again and returns you to the command menu. If monitoring is not suspended, no action occurs.

The monitor file can be either a .TEXT file or a data file. If it is a .TEXT file, you can use the editor to alter it, but only if the monitoring has not recorded special characters that the editor does not allow.

The M(onitor command itself can never be recorded in a monitor file.

R(un

On the menu: R(un

This command executes the current work file. If there is no current code file in the work file, the R(un command calls the compiler, and if the compilation is successful, runs the resulting code. If there is no work file at all, R(un calls the compiler, which then displays a request for the name of a text file to compile.

U(ser Restart

On the menu: U(ser Restart

This command causes the last program executed to be executed over again, with all file parameters equal to previous values. U(ser restart cannot restart the compiler or assembler. It is useful for multiple runs of your program.

X(ecute

On the menu: X(ecute

This command executes a program. It displays the following prompt:

```
Execute what file?
```

You should respond with an execution option string. In the simplest case, this string contains nothing but the name of a code file (program) to be executed.

If the code file cannot be found, the message:

```
No file <file name>
```

is displayed. If the program requires assembled code which has not been linked, the message:

```
Must L(ink first
```

is displayed. If the code file contains no program (that is, all its segments are unit or segment routines), the message:

```
No program in <file name>
```

is displayed.

If the execution option string contains only option specifications, they are treated as described under Execution Option Strings at the end of this section. If the string contains both option specifications and a code file name, the options are handled first and then the code file is executed, unless one of the errors named in the preceding paragraph occurs.

The X(ecute command is commonly used to call programs that have already been compiled. You may also use it to simply take advantage of the execution options.

The code file must have been created with a .CODE suffix, even if its name has subsequently been changed.

Execution-Option Strings

The X(ecute command allows you to specify some options that modify the system's environment. These include redirecting input and output, changing the default prefix, and changing the default library text file. These options are available from within programs as well as from the X(ecute command at the keyboard.

All of these options are specified by means of execution-option strings. An execution-option string is a string that contains (optionally) one file name followed by zero or more option specifications. An option specification consists of one or two letters followed by an equal sign =, which is possibly followed by a file name or literal string.

The following list contains the possible execution options with a summary of their uses.

L — change the default library text file
P — change the default prefix
PI — redirect program input
PO — redirect program output
I — redirect system input
O — redirect system output

Library text files are described in *UCSD p-System Program Development*. Prefixes are covered in this manual in Chapter 3, File Management, and I/O redirection is explained below.

You may use capital or lowercase letters with execution options. Several different execution options may be entered at a single time. They must be separated by one or more spaces. There may be a single space between the equal sign (=) and the following file name or string.

If you are executing a program, you must specify the name of the program to be executed *before* specifying any execution options. (These execution options can be specified in any order.)

The following items define the order in which execution options are actually performed.

1. Change the prefix if the P= option is present.

2. Change the library text file if the L= option is present.

3. Perform the I/O redirections (if any are present, the order of redirection options is irrelevant).

4. Execute the file if specified.

The execution options are described in the following paragraphs. They may be called by using the X(ecute command. Reduction from within a user program may be accomplished through procedures in a unit called COMMANDIO.

You can change the default prefix with the P= execution-option string. After this is done, all file names that do not explicitly name a volume are prefixed by the default prefix. This is equivalent to using the P(refix command in the filer.

To change the default prefix, press the X select the X(ecute command. Enter p = disk2 and press the **RETURN** key. The prefix is now DISK2:.

You can change the default user library text file in the same way. The library text file is a file that contains the names of your libraries. When you run a program with separately compiled units, the system searches for them first in the files named in the library text file and then in *SYSTEM.LIBRARY. When the system is booted, the default library text file is *USERLIB.TEXT. (This is all covered in *UCSD p-System Program Development.*)

To change the default library text file, press the X key to select the X(ecute command. Enter L = mylib to make the file MYLIB.TEXT the new default library text file.

Enter prog l = mylib to make the file MYLIB.TEXT the new library text file and execute the file PROG.CODE.

Redirection

The following execution-option strings control redirection:

PI = <file name>
PI = <string>
PO = <file name>
I = <file name>
I = <string>
O = <file name>

PI = <file name> or <string>

Redirects program input. PI = <file name> causes the input to a program to come from the file named. PI = <string> causes the input to a program to come from the program's scratch input buffer and appends the string given to the scratch input buffer (scratch input buffers are discussed in the following paragraphs).

PI = overrides any previous input redirection. Likewise, PO = overrides any previous output redirection. Using PI = (PO =) without a file name makes program input (output) the same as System input (output).

PO = <file name>

Redirects program output. PO = <file name> causes program output to be sent to the file named.

I = <file name> or <string>

Redirects system input. I = <file name> causes system input to come from the file named. I = <string> causes system input to come from the system's scratch input buffer, and appends the string to the scratch input buffer. Scratch buffers are described in the following paragraphs.

O = <file name>

Redirects system output. O = <file name> causes system output to be sent to the file named.

Like PI =, I = overrides any previous I =; and like PO =, O = overrides any previous O =. Using I = without a file name resets system input to CONSOLE:. Using O = without a file name resets system output to CONSOLE:.

For PI = <file name> and I = <file name>, the <file name> may specify either a disk file or an input device that sends characters. If the file is a disk file, redirection ends at the end of the file, and the system performs the equivalent of an input redirection with no file name, thus resetting input. If the file is a device, redirection continues until you explicitly change it. This allows you to control the system from a remote port (such as REMIN:).

For PO = <file name> and O = <file name>, the <file name> may specify either a disk file or an output device that receives characters. If the file is a disk file, it is named literally as shown; that is, to make it a text file, you must explicitly type .TEXT. Whenever output redirection is changed, the file is closed and locked.

For PI = <string> and I = <string>, the <string> may be any sequence of characters enclosed in double quotes (''). A comma within the string indicates a carriage return. Any double quote embedded in the string must be entered twice.

When input is redirected to a string, that string is placed in a first-in-first-out queue called the scratch input buffer. Anything that already exists in the scratch input buffer is read before the quoted string. The p-System has an area of memory devoted to its scratch input buffer. A program has a separate scratch input buffer of its own. If there is nothing already in the scratch buffer, it is as if input is taken immediately from the string itself.

If you redirect input to come from both a file and a scratch input buffer, the scratch buffer is used first.

Program redirection ends when the program terminates. If there are still characters in the program's scratch input buffer, they are not used.

System redirection ends when the system terminates with a halt or a run-time error. An I(nitialize command does not alter system redirection. The system's scratch input buffer is lost when system redirection terminates.

NOTE

The redirection applies only to high-level I/O operations, such as WRITELN and READLN in Pascal. Lower-level I/O operations, such as UNITREAD and UNITWRITE, are NOT intercepted, thus, cannot be redirected. Also, BLOCKREAD and BLOCKWRITE are not redirected. This means that if you redirect a program which uses any of these operations, they will not be redirected.

Redirection also cannot affect calls in the following form because these calls do not involve the standard input and output files:

```
REWRITE(MY_FILE,'CONSOLE:');
WRITE(MY_FILE, LOTS_OF_TEXT)
```

Here is a simple example of redirecting the system input to a string:

```
Execute what file? I="FL*,Q"
```

This causes the p-System to enter the filer (F), list the directory on the boot disk (L*,), remember that comma means <return>, and Q(uit the filer (Q).

To redirect program input to the file IN (which might have been created using M(onitor), and program output to the file OUT, for a program called PROG.CODE; press the X key to call the X(ecute command and respond:

```
Execute what file? PROG PI=IN PO=OUT
```

To stop system input redirection, enter I = .

If you enter:

```
PO = storeme.text  PI =  I = "fgRUNME,qr" P = WORK2
```

The p-System performs these actions:

- Makes the default prefix WORK2:
- Redirects program output to the file WORK2:STOREME.TEXT
- Turns off program input redirection
- Follows the script "fgRUNME,qr":
  f: enter the filer;
  gRUNME,: G(et the work file
  WORK2:RUNME.TEXT and
  WORK2:RUNME.CODE
  (The comma acts as a carriage return.)
  q: Q(uit the filer;
  r: R(un the program WORK2:
  RUNME.CODE.
  (Note that its output has been
  redirected.)

The following entry does the same thing:

```
PO = storeme.text PI =  I = "fpWORK2:,gRUNME,qr"
```

3

File Management

INTRODUCTION

This chapter covers topics which are relevant to managing the files on your disks with the UCSD p-System on the Texas Instruments Professional Computer.

First, files and volumes are described in general. File and volume naming conventions are covered. Also, the different types of files and volumes are presented.

Second, the work file is introduced. This is a special *scratch pad* file that you may want to use if you plan to develop programs.

The filer is then introduced. The filer is the p-System's major file-handling facility. It allows you to view the files on a disk volume, move them around, remove them, and so forth. Its menu is introduced. Also, a more advanced feature called wild card characters is covered. These may be used, in conjunction with the filer's prompts, to work with several files at one time.

The next section describes how you can attempt to recover any files that you accidentally lose. If you inadvertently remove a valuable file, for example, the procedures outlined here should assist you in retrieving it.

Subsidiary volumes are covered next. Subsidiary volumes allow you to have two levels of file directory information. More files can be stored on a disk if you use subsidiary volumes.

User-defined serial volumes are then introduced. If your p-System is set up to use these, you can take advantage of extra serial I/O peripherals (such as extra terminals or printers).

Finally, the filer commands are described in detail.

FILE ORGANIZATION

A file is a collection of information that is stored on a disk and referenced by a file name. Each disk contains a directory that has the name and location of every file that resides on it. A disk directory may hold as many as 77 files. If you need more on a single disk (which can easily be the case if you are using large capacity hard disks), you can use subsidiary volumes. (Subsidiary volumes are described later in this chapter.)

A file may contain any sort of data and be organized in many ways. Depending on the type of file, which is usually indicated by the file name suffix, the system treats it in specific ways. For example, your files may contain text such as letters and memos, or they may contain executable code. The p-System recognizes these differences.

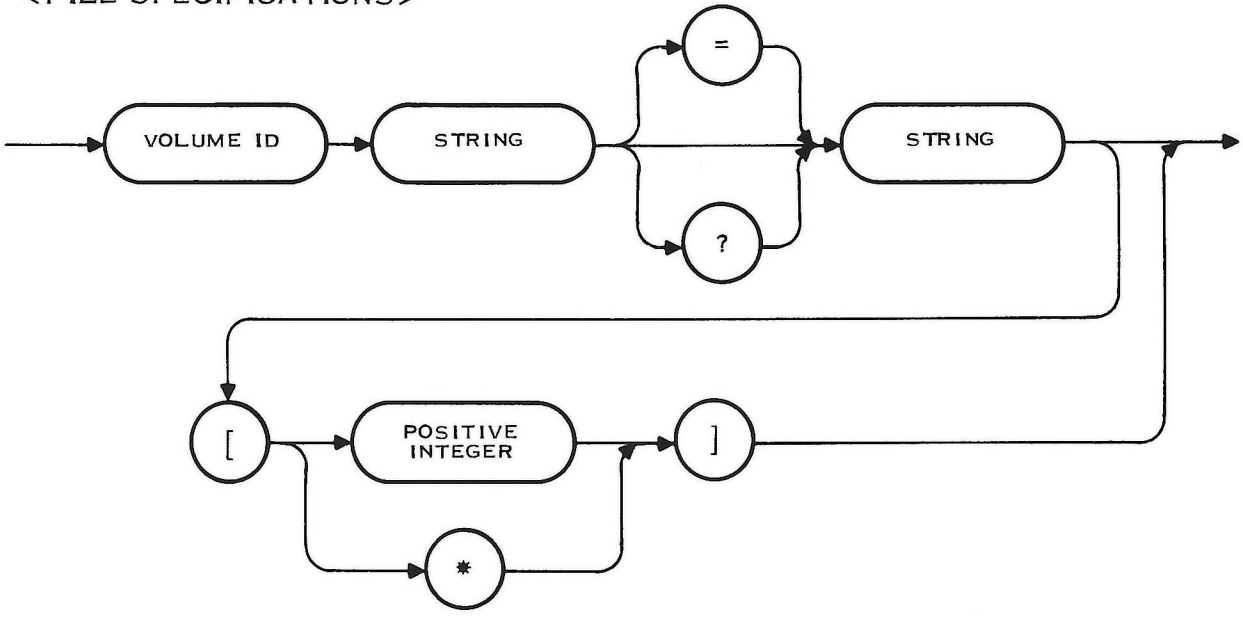
Disks (sometimes known as *storage volumes*) are also *volume names*. Sometimes disks are referenced by *device number* (described later). The term *volume ID* refers to a volume name or device number of a given storage volume.

The filer is a program that you start from the command menu. It provides a variety of commands that allow you to create, name and rename files, remove them, transfer them, print them, and so forth.

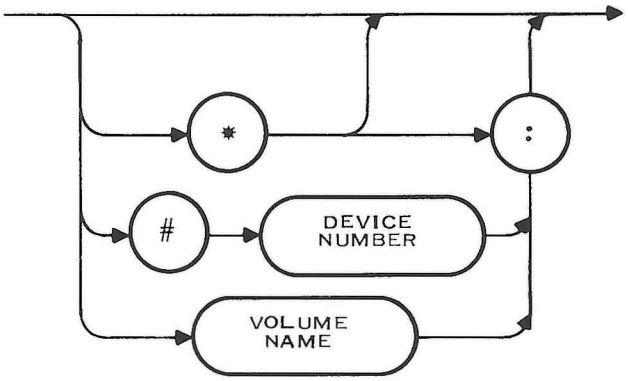
File and Volume Names

Many filer prompts require you to respond with a file or volume name. In fact, many p-System prompts, in general, require this. The following two figures show the technical syntax for file names and volume names.

<FILE SPECIFICATIONS>



2284050



2284051

The legal characters that you may use for file and volume names are:

- The alphabetic characters (A through Z)
- The numeric characters (0 through 9)
- Hyphen (-)
- Slash (/)
- Backslash (\ )
- Underline (_)
- Period (.)

File names can be, at most, 15 characters long. Here are some valid examples of file names:

```
A.FILE_NAME
MEMO.TEXT
PROGRAM/3.CODE
```

Here are some INCORRECT examples:

```
A,BAD,NAME
MORE_THAN_15_CHARS
#$*&-{
```

Volume names may be, at most, seven characters in length and are followed by a colon. Here are some correct examples:

```
VOLNAME:
VOL.2:
1234567:
```

Here are some INCORRECT volume names:

```
NOTCORRECT:
VOL$2:
SAY:HI:
```

Volumes may also be referenced by *device number*. A device number consists of a number sign (#) followed by a number, which is usually followed by a colon. Here are some examples:

```
#1
#1:
#4
#4:
#5
#9:
```

The colon is optional unless the device number is followed by a file name, as described below. (The colon is required after a volume name, however, to distinguish it from a file name.)

Disk drives usually have the device numbers #4 and #5, and sometimes also #9, #10, #11, #12, and even greater numbers. (Subsidiary volumes and *user-defined serial devices* may also use device numbers #9 and higher.) When you refer to a volume by device number, you are indicating the disk which happens to be in that drive at that time.

The asterisk (*) is shorthand for the volume ID of the system disk. The colon (:) is shorthand for the volume ID of the default disk (as described below). The system disk and default disk are equivalent unless the default prefix is changed. You can change it with the P(refix command. Sometimes the system disk is also called the boot disk.

Lowercase letters are translated to uppercase.

You may indicate the volume on which a file resides by using the volume name or device number (with colon) followed by the file name. Here are some examples:

```
MY.DISK:MY.FILE
DISK2:MY.FILE
#4:ANOTHER.TEXT
#5:PROGRAM.CODE
*BOOT.DISK.FILE
```

In the first two cases, the file MY.FILE is indicated, but on two separate volumes. The next two cases specify files on the disks in drives #4 and #5. The final example indicates a file on the system disk.

If you do not indicate a volume ID to go with your file name, that file is assumed to reside on the default disk. If, for example, the default disk is called MYDISK: and you answer a file name prompt with A.FILE, the p-System assumes (by *default*) that you are referring to MYDISK:A.FILE.

When a file is being created, its name may be followed by a size specification having the form [n], where n is an integer specifying the number of blocks that the file must occupy. For example, A.FILE.CODE[12] is made to occupy 12 blocks.

The following items describe some special cases:

- [0] This is equivalent to omitting the size specification. The file is created in the largest unused area.

- [*] The file is created in the second largest area or half the largest area, whichever is larger.

File Name Suffixes

User files are generally one of three types: program or document text, compiled or assembled program code, or data in a user-defined format. The suffix of a file name usually indicates its file type.

The following list summarizes the file suffixes:

.TEXT — Human-readable text, formatted for the editors.

.BACK — Same as a text file. Used for backup purposes.

.CODE — Executable code, either p-code or machine code.

.FOTO — A file containing one graphic screen image.

.BAD — An unmovable file covering a physically damaged area of a disk.

.SVOL — A file containing a subsidiary volume.

Data files, which contain data in a user-specified format, do not have any special suffix.

Here are some example file names which use these suffixes:

```
A.POEM.TEXT
DOCUMENT.BACK
A_PROG.CODE
FIGURE1.FOTO
BAD.00042.BAD
MYVOL.SVOL
A_DATA_FILE
```

.TEXT files contain human-readable information such as letters, poems, documents and so forth. .BACK files are backup files for text files. .TEXT and .BACK files contain a header page followed by the user-written text, interspersed with a few special codes. The header page contains internal information for the editors. The filer transfers the header page from disk to disk, but never from disk to an output device such as the PRINTER: or CONSOLE:.

All files created with a suffix of .TEXT have the header attached to the front. They are treated as text files throughout their lives.

The header page is two blocks long (1,024 bytes), with the remainder of the file also organized into two-block pages. A page contains a series of text lines, and is padded at the end with at least one NUL character.

Each line of text is terminated with an ASCII CR. A line may begin with a blank-compression pair consisting of an ASCII DLE followed by a byte whose value is 32 + n, where n is the number of characters to indent. Text lines are typically 0 through 80 characters long to fit on standard terminals.

.CODE files contain either compiled or assembled code. They begin with a single block called the segment dictionary, which contains internal information for the operating system and linker. Code files may also contain embedded information. Refer to the *USCD p-System Internal Architecture Manual* for detailed coverage of code files.

.FOTO files hold a graphics screen image and are used in conjunction with Turtlegraphics.

.SVOL files contain subsidiary volumes which are discussed later in this chapter.

.BAD files are stationary files used to cover physically damaged portions of a disk.

All of the filer commands (except G(et and S(ave) that reference specific files require the file name suffixes. G(et and S(ave supply these suffixes automatically to aid you in using the work file.

Devices and Volumes

A volume is any I/O device, such as the printer, the keyboard, or a disk. A storage device (sometimes known as a *block-structured* device) is one that can have a directory and files, usually a disk of some sort. A communication device (also known as a *nonblock-structured* device) does not have internal structure; it simply produces or consumes a stream of data. For example, the printer and console are communication devices.

The following table illustrates the reserved volume names and device numbers used to refer to the standard communication and storage devices.

Device Number	Device Name	Description
1	CONSOLE:	Screen and keyboard with echo
2	SYSTERM:	Screen and keyboard without echo
4	<disk name>:	The system disk
5	<disk name>:	The alternate disk
6	PRINTER:	A line printer
7	REMIN:	A serial input line
8	REMOUT:	A serial output line
9	<vol name>:	Additional disk drives, subsidiary volumes, and user-defined serial devices

The system distinguishes between storage and communication devices. Storage devices are usually disk drives. They contain removable volumes that have a directory and files. Internally, a volume is organized into randomly accessible, fixed-size areas of storage called blocks, each containing 512 bytes. Files may vary in size, but are always allocated an integral number of blocks.

Communication devices include printers, keyboards, and remote lines. They have no internal structure and deal with serial character streams. Communication devices may perform input functions, output functions, or both.

A device or a file may be either a source of data or a destination for data. Many of the filer's data transfer operations apply to devices as well as to files.

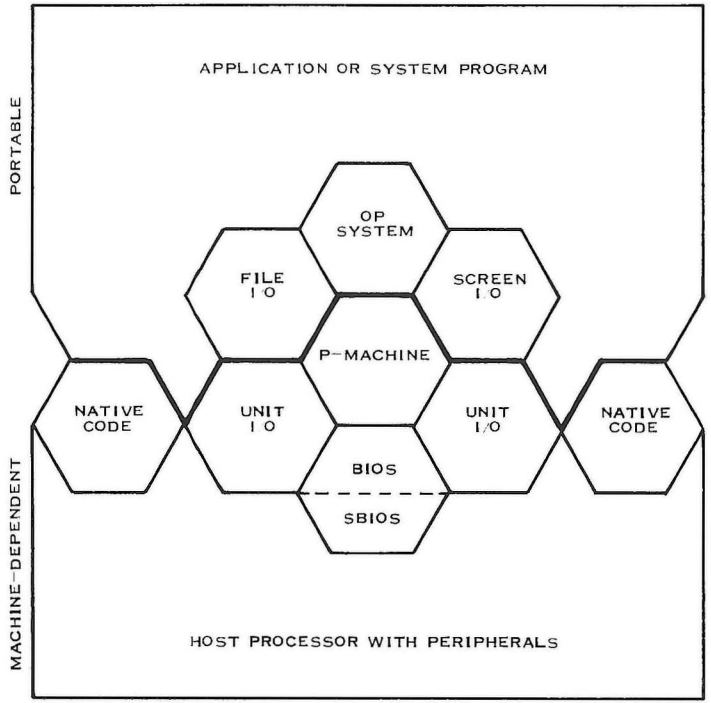
The name of a device that contains removable volumes, such as a diskette drive, is the name of the volume it contains at any given time. The number of that device never changes.

The name of a disk file includes, as a prefix, the disk on which it resides. The system always has one default prefix—when the system is first booted it is an asterisk (*), denoting the system disk—so that you need not type out the prefix every time a file is required.

For example, SYSTEM:SAVEME.TEXT and TABLES:SAVEME.TEXT name two different files on two different disks (both files are called SAVEME). These might also be specified as #4:SAVEME.TEXT and #5:SAVEME.TEXT. If you had changed the default prefix to TABLES:, then entering SAVEME.TEXT would be understood to mean TABLES:SAVEME.TEXT.

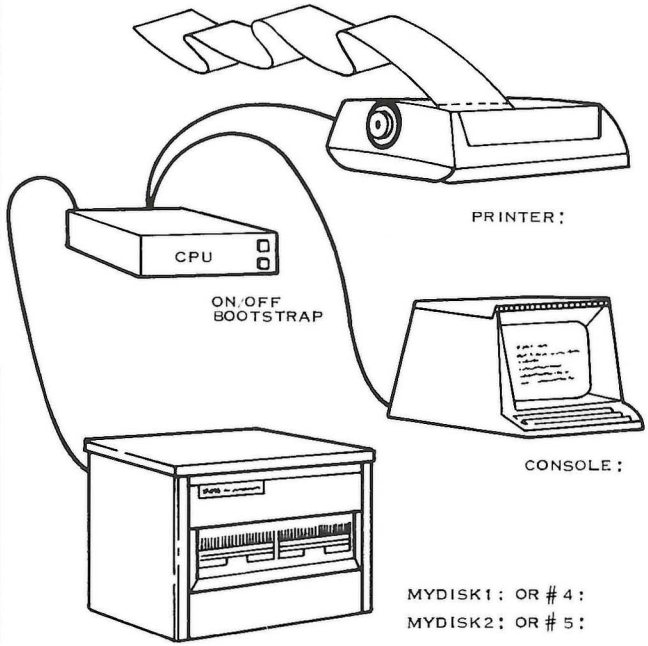
The following figure shows a typical hardware configuration with device names and a sketch of the operating system I/O interface.

UCSD P-SYSTEM I O HIERARCHY



2284052

A SAMPLE SYSTEM



WORK FILES

The work file is a scratchpad for creating and testing files. The work file is often stored temporarily in SYSTEM.WRK.TEXT and SYSTEM.WRK.CODE. These may be either newly created files or copies of existing disk files that have been designated as the work file.

Many system programs assume that you are working on the work file unless you specify otherwise. You may create the work file by designating existing files or by creating a new file with the editor.

Modifying the work file can cause temporary copies to be generated, which—until they are saved—are placed in the directory under the following names.

SYSTEM.WRK.TEXT
SYSTEM.WRK.CODE
SYSTEM.LST.TEXT

You can create SYSTEM.WRK.TEXT by leaving the editor if you use Q(uit U(pdate. Then a successful compile or run creates SYSTEM.WRK.CODE. If the compilation is successful, the R(un command goes on to immediately execute the code. The compiler may optionally create SYSTEM.LST.TEXT which is a compiled listing.

Whenever the editor alters a program contained in the file SYSTEM.WRK.TEXT, the R(un command recompiles it in order to update SYSTEM.WRK.CODE.

The filer can S(ave these files under permanent names. You can also use it to designate a new work file with the G(et command or to remove an old one with the N(ew command. The filer can also tell you W(hat your work file's name is.

USING THE FILER

Filer Menus

With the command menu displayed, press the F key to enter the filer. The system displays the following menu.

```
Filer: G(et, S(ave, W(hat, N(ew, L(dir, R(em, C(hng,
  T(rans, D(ate, ? [version]
```

Enter ?. The system then displays more filer commands:

```
Filer: Q(uit, B(ad-blks, E(xt-dir, K(rnch, M(ake,
  P(refix, V(ols, ? [version]
Filer: X(amine, Z(ero, O(n/off-line, F(lip-swap/lock
```

The individual filer commands are selected by entering the letter found to the left of the parenthesis. For example, S would call the S(ave command.

In the filer, answering a Yes/No question with any character other than Y or y constitutes a no answer. Pressing the ESC key returns you to the main filer menu.

Many commands display a prompt asking for a file or volume name. We have already discussed what file and volume names are. You can, of course, use a volume ID as part of a file name when responding to these prompts. In some cases, *either* a file or a volume may be indicated.

If you specify a file on a volume (or just a volume) that the filer cannot find, the system displays the following message:

```
No such vol on line
```

If two or more on-line volumes have the same name, the filer continuously displays a warning.

CAUTION

Although sometimes it may be necessary to have two volumes with the same name on-line at the same time, try to avoid this. You can confuse the p-System and accidentally destroy valuable information on one of the volumes.

Whenever a filer command requests a file specification, you may specify as many files as desired by separating the file specifications with commas and terminating the file list with the RETURN key. Commands operating on single file names read file names from the file list and operate on them until none are left.

Commands operating on two file names (such as C(hange and T(rans) take file specifications in pairs and operate on each pair until only one or none remains. If one file name remains, the filer displays a menu requesting the second member of the pair. If an error is detected in the list, the remainder of the list is flushed.

Wild Card Characters

Wild card characters allow the filer to perform its task on several files at a time. There are three wild card symbols: equal sign (=), question mark (?), and dollar sign ($).

The equal sign and question mark are used to specify subsets of the directory. The filer performs the requested action on all files meeting the specification.

The equal sign matches any string. For example:

```
=.TEXT
```

matches all of the following:

```
FILE1.TEXT
FILE2.TEXT
ANOTHER.TEXT
```

If a question mark is used in place of an equal sign, the filer requests verification before performing the function on each file watching the wild card specification. For example, if you want to R(emove some, but not all text files on a disk, you could use ?.TEXT and you are prompted for each file if you want it removed.

A wild card character specification must be of the form:

```
=
?
$=<string>
?<string>
<string>=
<string>?
<string>=<string>
<string>?<string>
```

These last two cases, where there is no string to match, is understood to specify every file on the volume. So entering = or ? alone causes the filer to perform the appropriate action on every file in the directory. Only one valid wild card character can occur in a specification.

The following paragraphs describe the use of the filer with wild card characters.

The following listing is the directory for volume DISK1.

```
TEMP1              6  1-Jan-83
OLD.TEXT           4  1-Jan-83
EXAMPLE1.CODE     10  1-Jan-83
EXAMPLE2.CODE      4  1-Jan-83
NEW.TEXT          12  1-Jan-83
TEMP2              5  1-Jan-83
TEMP.CODE          2  1-Jan-83
```

With the command menu displayed, press the F key to call the filer. Then press the R key to use the R(emove option. The system will display the following prompt:

```
Remove what file?
```

Enter TEMP= and press the **RETURN** key.

The system then displays the following listing:

```
DISK1:TEMP2          removed
DISK1:TEMP.CODE      removed
Update directory?
```

To verify and complete this operation, press the Y key. To stop the operation, press the N key. If you press the N key, the files will not be removed.

Using the same directory to list a specified set of files, press the F key (shown on the command menu) and then press the L key to use the L(ist option. The system will display the following prompt:

```
Dir listing of what vol ?
```

Enter =TEXT and press the **RETURN** key. The system will display the following listing:

```
OLD.TEXT     4     1-Jan-83
NEW.TEXT    12     1-Jan-83
```

The subset-specifying strings may not overlap. For example, EXAMPLE.C=CODE would not specify the file EXAMPLE.CODE, whereas EXA=CODE would be a valid specification.

In any file name pair, you may use the character $ to signify the same file name as the first name, perhaps with a different volume ID or size specification.

Press the F key (command menu) and then press the T key to select the T(ransfer option. The system will display the following prompt:

```
Transfer what file?
```

Enter #5:RE.USE.TEXT,*$ and press the **RETURN** key. The system now transfers the file RE.USE.TEXT on device #5 (a disk drive) to the system disk (*), which is also device #4. The name will not be changed. The system will display the following message:

```
WORKSET:RE.USE.TEXT->SYSTEM:RE.USE.TEXT
```

RECOVERING LOST FILES

When a file is removed, it is actually removed from the directory, not the disk. The information that it contained remains on the disk until another file is written over it (which could happen at any time, since the filer considers it usable space).

If a file is accidentally removed, be careful not to perform any actions (whether from the system or from your program) that write to the disk, since they might write over the lost file. The K(runch command is virtually certain to do this, so avoid it.

With the command menu displayed, press the F key to call the filer and then press the E key to use the E(xtended list command. The E(xtended list command then displays the names of files in the directory and any unused blocks that have once contained files. Sometimes, by looking at the size of unused areas and their location in the directory, you can tell where the lost file was located.

With the filer menu displayed, press the M key to use the M(ake command. You should then enter a file name and the size in blocks (enclosed in brackets) of the lost file.

To recover a lost file with the M(ake command, the size specification should match the size of the file that was lost. If you remember the size, or if the lost file took up all the space between two files that are still listed in the directory, recovery is easy.

The M(ake command creates a file (of the size that you specify) at the beginning of the first available location on the disk which is at least that large. To fill up any unused (and unwanted) space that precedes the location of the lost file, use the M(ake command to create dummy files. (Later, you may remove these *filler* files.)

The following is an example of a listing made using the E(xtend list command:

```
WORK:
SYSTEM.MISCINFO     1   1-Jan-83      6    512   Datafile
<UNUSED>            1                 7
SYSTEM.SYNTAX      14   1-Jan-83      8    512   Datafile
REM.WRK.CODE        4   1-Jan-83     22    512   Codefile
<UNUSED>           75                26
MYFILE.TEXT        20   1-Jan-83    101    512   Textfile
<UNUSED>          373               121
4/4 files<listed/in-dir>, 45 blocks used, 449 unused,
  373 in largest
```

MYFILE.CODE was four blocks long and was located just after MYFILE.TEXT. To create it, press the M key (filer menu) to use the M(ake command and enter FILLER[75]. This procedure fills up the 75 blocks of unused space on the disk. Next, using the M(ake command, create a file with the following specifications: MYFILE.CODE[4]. MYFILE.CODE is created (once again) immediately following MYFILE.TEXT. Finally, use the R(emove command to delete FILLER from the directory.

The following extended listing results from this procedure.

```
WORK:
SYSTEM.MISCINFO     1  1-Jan-83      6   512  Datafile
<UNUSED>            1                 7
SYSTEM.SYNTAX      14  1-Jan_83       8   512  Datafile
REM.WRK.CODE        4  1-Jan-83      22   512  Codefile
<UNUSED>           75                26
MYFILE.TEXT        20  1-Jan-83     101   512  Textfile
MYFILE.CODE         4  1-Jan-83     121   512  Codefile
<UNUSED>          369               125
5/5 files<listed/in-dir>, 49 blocks used, 445 unused,
  369 in largest
```

NOTE

To X(ecute a code file, you must have created it with a .CODE suffix. (Later, you may change the code file name.) If you lose a code file that does not have a .CODE suffix (for example, SYSTEM.FILER) you must recreate the file with a .CODE suffix (for example, FILER.CODE) and then again change the name back to SYSTEM.FILER. If you do not do this, the recreated file will not be executable.

The RECOVER utility, described in Chapter 5, can help you find files when you cannot remember or determine where they were located on the disk. RECOVER scans the directory for entries that look valid. If that search does not yield the desired file, RECOVER attempts to read the entire disk looking for areas that resemble files and asks you if you want them re-created.

Another alternative is to use the PATCH utility to manually search through the disk. Once the file has been found, use M(ake to create the proper directory.

If a directory entry seems erroneous or confusing, you may use the PATCH utility to examine the exact contents of the directory. (Refer to *UCSD p-System Program Development*.)

Duplicate Directories

It is often easiest to recover a disk when the disk contains a duplicate directory. The main directory spans blocks 2 to 5 on a disk. If a duplicate directory is present, it spans blocks 6 to 9. Every time the directory is altered, the duplicate directory is updated as well, thus providing a convenient backup.

However, if a file is accidentally removed from a directory, it will also be removed from a duplicate directory and will not be recoverable.

If a directory is corrupted on a disk that has a duplicate directory, you may use the COPYDUPDIR utility to simply move the duplicate directory to the location of the standard disk directory. Sometimes this is all that is required to recover a disk.

There are two ways to place duplicate directories on a disk. The first is to instruct the Z(ero command to do this when you are initializing a disk's directory. When the prompt `Duplicate dir?` appears, press the `Y` key for yes. This prompt also appears in the M(ake command when you are creating subsidiary volumes. In this case, you can create a duplicate directory for the subsidiary volume if you wish.

If you are already using a disk that contains only one directory, you can use the MARKDUPDIR utility to create a duplicate directory (without having to zero the volume). However, be careful when using this utility. Blocks six to nine of the disk—the location of the duplicate directory—must be unused; if not, file information will be lost.

If a directory is lost, and no duplicate directory was present, use the RECOVER utility as previously described.

CAUTION

You will destroy the directory if you use the filer E(xtended list or L(ist commands and specify an optional output file as a disk volume *without* a file name. (The listing is written on top of the directory.)

EXAMPLE:

The L(ist directory prompts:

```
Dir listing of what vol ?
```

Response:

```
MYDISK:, MYDISK: RETURN
```

Response:

```
MYDISK:,: RETURN
```

Either of these responses cause the first few blocks (approximately six) of MYDISK: to be overwritten with a listing of the directory of MYDISK: if the prefix is MYDISK.

Response:

```
MYDISK:, DISK2: RETURN
```

This causes the directory of DISK2: to be overwritten.

In the latter case, you must use the disk recovery methods already described. In the first two cases, recovery is not so difficult, even if there was not a duplicate directory, since the MYDISK: directory has been overwritten with what is essentially a copy of itself.

First, get a copy of the directory listing of MYDISK:. (If MYDISK: was the system disk, you must boot another system.) Use the filer to T(ransfer MYDISK: to an output device: PRINTER:, REMOUT:, or CONSOLE:.

Generate hard copy of the directory and then use the filer to Z(ero MYDISK:. The Z(ero command will not alter the contents of MYDISK:, only the directory itself. Now use the M(ake command to remake all of the files on the disk (as described in the preceding paragraphs).

SUBSIDIARY VOLUMES

The purpose of subsidiary volumes is to provide two levels of directory hierarchy and to expand the p-System's ability to use large storage devices such as Winchester disk drives. Currently, p-System disk volumes contain a 4-block directory located in blocks 2 through 5. The rest of the disk contains the actual files described in the directory. The size of the directory allows for a maximum of 77 files to reside on the corresponding disk image.

Subsidiary volumes are virtual disk images that actually reside within a standard p-System file. The disk that contains one of these files is called the principal volume. Each subsidiary volume may contain up to 77 files.

A subsidiary volume appears in the directory of the principal volume as a file. Subsidiary volume file names can have a maximum of seven characters and must be followed by the suffix .SVOL. The following listing is an example.

```
MAIL.SVOL
TESTS.1.SVOL
DOC_B.SVOL
```

The subsidiary volume disk image resides within the actual .SVOL file. The directory format and file formats are the same as for any other p-System disk volume. The volume name of the subsidiary volume is that portion of the corresponding file name that precedes the .SVOL. For example, the three preceding files would contain the following subsidiary volumes.

```
MAIL:
TESTS.1:
DOC_B:
```

Creating and Accessing SVOLs

To create a subsidiary volume, use the filer M(ake command and the file name suffix, .SVOL. As with any other file the M(ake command creates, the subsidiary volume occupies:

1. All of the largest contiguous disk area if created as follows:

   ```
   Make what file? DOCS.SVOL
   ```

2. Half of the largest area or all of the second largest area, whichever is larger, if created as follows:

   ```
   Make what file? DOCS.SVOL[*]
   ```

3. A specified number of blocks, in the first area large enough to hold that many blocks, if created as in the following examples:

   ```
   Make what file? DOCS.SVOL[200]
   Make what file? DOCS.SVOL[1500]
   ```

After you enter the .SVOL file name, the system sometimes displays this prompt:

```
Zero subsidiary volume directory?
```

If you respond by pressing the Y key, the directory of the new subsidiary volume is zeroed. If you press the N key, the directory is not zeroed, and any files that may have existed on a previous subsidiary volume in the same location reappear within the directory. In both cases, the number of blocks indicated within the directory always correspond to the size of the actual .SVOL file. If this prompt is not displayed, then there was not a previous subsidiary volume directory where you are creating the current .SVOL file. In this case, the new subsidiary volume is automatically zeroed.

The next prompt which is almost always displayed is:

```
Duplicate dir?
```

You should respond by pressing the Y key if you want a duplicate directory to be maintained on the subsidiary volume, and pressing the N key otherwise. Duplicate directories were covered earlier under Recovering Lost Files.

Subsidiary volumes may not be nested. That is, an .SVOL file may not be created within another .SVOL file.

When you create a subsidiary volume, it is automatically placed on-line. You may then access and use it like any other p-System volume. The filer command, V(olumes, then displays a listing which indicates that the new volume is on-line and shows its corresponding device number; for example, #13:.

You may use either volume name or the device number when referencing the subsidiary volume. You may now place files on the new subsidiary volume, and all of the applicable file commands may reference it.

Mounting and Dismounting SVOLs

A mounted subsidiary volume is subtly different from an on-line subsidiary volume.

To identify a subsidiary volume as mounted means that the p-System knows the volume exists and sets aside a device number for it; for example, #13:. You must mount a subsidiary volume before you can use it. While it is mounted, only that specific subsidiary volume corresponds to that device number.

A subsidiary volume stays mounted until you dismount it. Once mounted, it is on-line any time its principal volume is in the disk drive. It is off-line when the principal volume has been removed from the disk drive.

CAUTION

There is a danger of confusing the system if two principal volumes each contain a subsidiary volume in the same location with the same name. This might easily be the case where backup disks are used. If these principal volumes are swapped in and out of the same drive, and the similar subsidiary volumes are accessed, the filer may become confused in the same way that it can when any two on-line volumes have the same name.

CAUTION

If you write programs, be careful when using low-level I/O routines (like UNITWRITE) with subsidiary volumes. If you remove a principal volume from a disk drive and insert another disk, these low-level routines have no way of knowing that the subsidiary volumes that were mounted on the original disk are no longer present. Under these circumstances, doing a UNITWRITE to absent subsidiary volumes overwrites data on the disk presently occupying the disk drive.

When you boot the p-System, all of the on-line disks are searched for .SVOL files. The corresponding subsidiary volumes are then mounted. The same process occurs whenever the p-System is initialized (by the I(nitialize command or after an execution error).

The booting or initializing process mounts as many subsidiary volumes as it finds as long as there is room in the p-System unit table. If the unit table becomes full, no more subsidiary volumes are mounted, and no warning is given. (The maximum number of subsidiary volumes is discussed a little later.)

After booting or initializing, if you place a new physical disk on-line, you must manually mount any subsidiary volumes contained on it if you want to access them.

To mount or dismount subsidiary volumes, use the O(n/off-line command. From the main filer menu press the O key. The system will display the following menu:

```
Subsidiary Volume: M(ount, D(ismount, C(lear
```

Press the M key. The system displays this prompt:

```
Mount what vol ?
```

To dismount a subsidiary volume, press the D key. The system displays this prompt:

```
Dismount what vol ?
```

To dismount all the subsidiary volumes that are currently on-line, press the C key.

Suppose that a principal volume, P__VOL:, contains the following files:

```
P__VOL:
  FILE1.TEXT
  FILE1.CODE
  VOL1.SVOL
  FILE2.TEXT
  FILE2.CODE
  DOC1.SVOL
  FUN.SVOL
```

To mount subsidiary volumes on P__VOL:, you can respond to the mount prompt with the file name, as in the following examples:

```
Mount what vol? VOL1.SVOL RETURN

Mount what vol ? VOL1.SVOL, FUN.SVOL RETURN

Mount what vol ? P__VOL:= RETURN

Mount what vol ? #5:=RETURN
```

The first example mounts VOL1:; the second mounts VOL1: and FUN:; the third mounts all three subsidiary volumes on P__VOL:; and the fourth example mounts all subsidiary volumes on the disk in drive #5:.

To dismount any of these volumes, you can respond to the dismount prompt with the VOLUME ID as in the following examples:

```
Dismount what vol ? #14:

Dismount what vol ? VOL1: RETURN

Dismount what vol ? VOL1:, DOC1:, FUN: RETURN
```

The first example dismounts the subsidiary volume associated with device number #14. The second example dismounts VOL1:, and the third example dismounts three subsidiary volumes.

There is a maximum number of subsidiary volumes that you may mount at one time. You can set this number, which is subject to memory constraints and tradeoffs. The maximum number of subsidiary volumes is a field in SYSTEM.MISCINFO and is configured using the SETUP utility (which is covered in Chapter 5).

NOTE

If you C(hange either the name of a subsidiary volume or the name of the corresponding .SVOL file, you must change them both to the same name. For example, if you want to change either of these:

MYVOL.SVOL
MYVOL:

You should C(hange both of them in the same way:

NEWNAME.SVOL
NEWNAME:

Installation Information

It is very simple to install the subsidiary volume facility if you use the SETUP utility to set MAX NUMBER OF SUBSIDIARY VOLS to the smallest convenient value. This will be the maximum number of subsidiary volumes that are allowed to be mounted at one time. (Each additional subsidiary volume requires a few extra bytes within the p-System's unit table. This is why you should keep this number as small as possible.) When you have set this field, the subsidiary volume facility is available.

FILER COMMANDS

This section describes filer commands and gives examples of their use. Activities are listed in alphabetical order with each new command beginning on a new page.

B(ad Blocks

On the menu: B(ad-blks

This command scans a volume's data blocks and detects areas that are apparently bad for some physical reason (magnetic damage, fingerprints, warping, dirt, and so on).

This command requires you to enter a volume ID. The specified volume must be on-line.

Prompt:

```
Bad block scan of what vol?
```

Response:

```
<volume ID>
```

Prompt:

```
Scan for 640 blocks ? <y/n>
```

Press the Y key for yes to scan for the entire length of the disk. To check a smaller portion of the disk, press the N key. The system will then display a prompt requesting the number of blocks which the filer should scan.

The system checks each block on the indicated volume for errors and lists the number of each bad block. Bad blocks can sometimes be fixed or marked (see X(amine).

C(hange

On the Menu: C(hng

This command changes file or volume name.

C(hange requires two file names. The first name specifies the file or volume name to be changed, the second entry specifies the name it is to be changed to. The first entry is separated from the second entry by either pressing the **RETURN** key or the comma key (,). Any volume name information in the second file specification is ignored since only the name in the volume directory is changed. Size specification information is also ignored.

The following example shows how to change file or volume names. The example file F5.TEXT resides on the volume occupying device #5:

Prompt:

```
Change what file?
```

Response:

```
#5:F5.TEXT,NEWNAME
```

The preceding procedure changes the name in the directory from F5.TEXT to NEWNAME. File types are originally determined by the file name; however, the C(hange command does not affect the file type. In the above case, NEWNAME is still a text file.

On the other hand, a response of

```
#5:F5 = ,NEWNAME =
```

preserves the .TEXT suffix.

Wild card character specifications are legal in the C(hange command. If you use a wild card character in the first file specification, then you must use a wild card character in the second file specification. The subset-specifying strings in the first file specification are replaced by the analogous strings (called replacement strings) given in the second file specification.

The filer will not change the file name if the change would make the new file name too long, that is, more than 15 characters.

EXAMPLE:

Given a directory of example disk DISK1: containing the following files:

```
EXAMPLE.TEXT
MAIL.TEXT
MAIL.CODE
MAKE.TEXT
```

Prompt:

```
Change what file?
```

Response:

```
DISK1: MA = TEXT RETURN
```

Prompt:

```
Change to what?
```

Response:

```
XX = WHAT
```

This causes the filer to report:

```
DISK1:MAIL.TEXT --> XXIL.WHAT
DISK1:MAKE.TEXT --> XXKE.WHAT
```

The subset-specifying strings may be empty, as may the replacement strings. The filer considers the file specification equal sign (=) (where both subset-specifying strings are empty) to specify every file on the disk. Responding to the C(hange prompt with =,Z=Z causes every file name on the disk to have a Z added at the front and back. Responding to the prompt with Z=Z,= replaces each terminal and initial Z with nothing.

EXAMPLE:

Given the file names:

```
THIS.TEXT
THAT.TEXT
```

Prompt:

```
Change what file?
```

Response:

```
T=T,=
```

The result would be to change THIS.TEXT to HIS.TEX, and THAT.TEXT to HAT.TEX.

You may also change the volume name by specifying a volume ID to be changed and a new volume ID.

EXAMPLE:

Prompt:

```
Change what file?
```

Response:

```
DISK1:,DISK2:
```

Causes the filer to report:

```
DISK1:              --> DISK2:
```

D(ate

On the menu: D(ate

This command lists the current p-System date and enables you to change it if you want.

```
Prompt:  Date Set:<1..31>-<JAN..DEC>-<00..99>
Today is 31-Dec-82
New date?
```

You may enter the correct date in the format given. After pressing the **RETURN** key, the new date is displayed. Pressing only a return does not affect the current date. The hyphens are delimiters for the day, month, and year fields, allowing you to affect only one or two of these fields.

For example, you can change only the year by entering --83, only the month by entering - Jan, and so on. You can spell out the name of the month entirely, but the filer will truncate it.

The most common input is a single number, which is interpreted as a new day. For example, if the date shown is the 1st of January, and today is the 2nd, you enter 2 and press the **RETURN** key; this procedure changes the date to the 2nd of January. The day-month-year order is required.

The p-System's date is associated with any files which are created or modified during the current session. Thus, the individual files may have different dates. These dates are displayed when the directory is listed.

The p-System's date is saved in the directory of the system disk. The date remains the same until you change it by using the D(ate command.

E(xtended List

On the menu: E(xt-dir

This command lists the directory in more detail than the L(dir command. See L(dir for more information.

All files are listed with their block length, last modification date, the starting block address, the number of bytes in the last block of the file, and the file type. The unused areas are also displayed. All wild card character options and prompts are used in the same way as the L(dir command.

Since this command shows the complete layout of files and unused space on the disk, it is useful in conjunction with the M(ake command. (You can see where files may be created.)

Often, an E(xtended list is too long to fit on one screen. In this case, the filer displays one full screen and then prompts:

```
Type <space> to continue
```

You should press the space bar to list the rest of the directory. Press the ESC key to abort the listing.

EXAMPLE:

Here is a sample extended listing:

```
MYDISK:
FILERDOC2.TEXT    28    1-Jan-83    6     512   Textfile
MEMO.CODE         18    1-Jan-83    34    512   Codefile
<UNUSED>          10                52
SCHEDULE           4    1-Jan-83    62    512   Datafile
HYTYPER.CODE      12    1-Jan-83    66    512   Codefile
STASIS.TEXT        8    1-Jan-83    78    512   Textfile
LETTER1.TEXT      18    1-Jan-83    86    512   Textfile
ASSEMDOC.TEXT     20    1-Jan-83   104    512   Textfile
FILERDOC1.TEXT    24    1-Jan-83   124    512   Textfile
<UNUSED>         200                148
STASIS.CODE        6    1-Jan-83   348    512   Codefile
<UNUSED>         154                354
10/10 files <listed/in-dir>, 138 blocks used,
   364 unused, 200 in largest
```

F(lip Swap/Lock

On the menu: F(lip swap/lock

This command can facilitate the use of the filer on systems that have enough memory.

The Pascal code that makes up the filer is divided into several segments. Not all of the segments are needed in main memory at the same time. By removing unnecessary segments from memory, more memory space is available for the filer to perform its tasks. For example, a transfer is much more efficient when there is a large buffer area available in memory. Furthermore, on some machines, there just is not enough memory space to contain the entire filer.

However, allowing the filer to have nonresident segments requires that the disk containing SYSTEM.FILER be accessed whenever a nonresident segment is needed. This can be inconvenient on one-drive systems. It would be more convenient to do the following: Enter the filer, remove the system disk, if desired, and perform any combination of L(isting, disk-to-disk T(ransferring, K(runching, and so on, without having to replace the system disk at frequent intervals.

In the first mode, the filer segments are memswapped in and out of memory; in the second mode, they are memlocked into memory. The F(lip swap/lock command allows you to choose the mode the filer will use. Upon entering the filer, the initial state is always the memswapped state. Pressing the F key acts as a toggle between the memswapped and memlocked states.

For example, if you enter the filer and press the F key twice, the system displays two messages similar to these:

```
Filer segments memlocked [9845 words]
Filer segments swappable [13918 words]
```

The number of available 16-bit words is given so that you will have an idea of how much space is left for the filer to perform its functions. There is usually less space available in the memlocked mode. If the machine does not have enough space to memlock the filer segments, you receive a message so. (If there are not at least 1,500 extra words available, the filer will not allow the memlock option.)

G(et

On the menu: G(et

This command designates a text and/or code file as the work file.

The entire file specification is not necessary. If the volume ID is not given, the default disk is assumed. Wild cards characters are not allowed, and the size specification option is ignored.

EXAMPLE:

Given the directory:

```
MEMO.TEXT
PRINT.CODE
PROG.TEXT
PROG.CODE
```

Prompt:

```
Get what file?
```

Response:

```
PROG
```

The filer responds with the following message because both text and code files exist.

```
Text & Code file loaded
```

If you enter PROG.TEXT or PROG.CODE, the result is the same. Both text and code versions are loaded. If only one of the versions exists, as in the case of MEMO, then that version is loaded, regardless of whether you requested text or code. For example, entering MEMO.CODE in response to the prompt generates the message: Text file loaded.

Using the compiler, editor, assembler on a work file may cause the files SYSTEM.WRK.TEXT and/or SYSTEM.WRK.CODE to be created as part of the work file. The SYSTEM.WRK files disappear when you use the S(ave command. If you reboot the p-System before using the S(ave command, the p-System forgets the name of the work file. In this case, the p-System does not know what files the SYSTEM.WRK files were derived from.

K(runch

On the menu: K(rnch

This command moves the files on a volume together so that the unused space is consolidated into one large area.

K(runch first displays a prompt asking for the name of a volume. It then asks if it should move the files from the end of the volume toward the beginning. If you answer yes to this question, K(runch leaves all files at the front of the volume, and one large unused area at the end. If you answer no to this prompt, K(runch asks at which block the file movement should start. Doing a K(runch from a block in the middle of the volume leaves a large unused area in the middle of the volume, with files clustered toward either end (as space permits). Doing a K(runch from the beginning of a volume leaves the files at the end and the unused space at the beginning.

As each file is moved, its name is displayed on the console.

If the volume contains a bad block that has not been marked (see B(ad and X(amine), K(runch may move a valuable file on top of it. That file is then beyond recovery. You should scan for bad blocks with the B(ad command before using the K(runch command unless all files are also backed up on a different volume.

If the K(runch command must move SYSTEM.PASCAL or SYSTEM.FILER on the system disk, it then displays a prompt which asks you to re-boot the system.

EXAMPLE:

Prompt:

```
Crunch what vol?
```

Response:

```
MYDISK:
```

If MYDISK: is on-line, K(runch displays a prompt similar to this:

```
From end of disk, block 640 ? (y/n)
```

The 640 indicates the last block on your volume and may be different for your disks. To start the crunch, from this location, press the Y key. To start the crunch at another location, press the N key and this is displayed:

```
Starting at block # ?
```

Enter the block number at which the crunch should begin.

The contents of subsidiary volumes can be crunched just like any other volume.

L(ist Directory

On the menu: L(dir

This command lists the files in a disk directory or some subset of them. Usually the listing is displayed on the console, but you can direct it to a file or to a communications volume, such as PRINTER:.

Each file name is followed by the file length, in blocks (a block is 512 bytes), and the date of its last modification.

When you select L(ist directory, this prompt is displayed:

```
Dir listing of what vol?
```

You can respond to this with a storage volume name. The directory of this volume is then listed. If you want, you can follow the volume name with a file name or wild card character expression for multiple file names. In this case, the single file or the subset of the directory indicated by the wild card character expression is listed.

You can, if you want, send the listing to a communications volume (such as PRINTER:) or a file (such as LIST.TEXT). To do this, use a comma after you indicate the volume to be listed. Following the comma, type in the destination for the listing.

If the directory listed is too long to fit on one screen, the filer lists as much of it as it can and then displays the following prompt:

```
Type <space> to continue
```

Pressing the space bar causes the rest of the directory to be listed; pressing the ESC key halts any further listing.

EXAMPLE:

To list MYDISK:, select L(ist directory and respond like this:

Prompt:

```
Dir listing of what vol?
```

Response:

```
MYDISK:
```

Here is the listing of MYDISK:

```
MYDISK:
FILER1.TEXT          38    1-Jan-83
PRINT.CODE            5    1-Jan-83
FILE2.TEXT           22    1-Jan-83
MEMO.TEXT            30    1-Jan-83
FILE3.TEXT           25    1-Jan-83
5/5 files <listed/in-dir>, 120 blocks used,
  100 unused, 100 in largest
```

The bottom line of the display informs you that: 5 files out of 5 files on the disk have been listed, 120 blocks have been used, 100 blocks remain unused, and the largest area available is 100 blocks.

The following example is a list directory transaction involving wild card characters:

Prompt:

```
Dir listing of what vol ?
```

Response:

```
MYDISK:FIL=TEXT
```

The system displays the following listing:

```
MYDISK:
FILE1.TEXT          38    1-Jan-83
FILE2.TEXT          22    1-Jan-83
FILE3.TEXT          25    1-Jan-83
2/5 files <listed/in-dir>, 85 blocks used,
  100 unused, 100 in largest
```

The following example is a list directory transaction that involves writing the directory subset to a device other than CONSOLE.

```
Dir listing of what vol ?
```

Response:

```
MYDISK:FIL=TEXT,PRINTER:
```

The system prints the following listing:

```
MYDISK:
FILE1.TEXT          38    1-Jan-83
FILE2.TEXT          22    1-Jan-83
FILE3.TEXT          25    1-Jan-83
2/5 files <listed/in-dir>, 85 blocks used,
  100 unused, 100 in largest
```

EXAMPLE:

The following example is a list directory transaction that involves writing the directory subset to a file:

Prompt:

```
Dir listing of what vol ?
```

Response:

```
MYDISK:FIL=TEXT,#5:LIST.TEXT
```

The system creates the file LIST.TEXT on the disk in drive #5. LIST.TEXT contains this listing:

```
MYDISK:
FILE1.TEXT          38    1-Jan-83
FILE2.TEXT          22    1-Jan-83
FILE3.TEXT          25    1-Jan-83
2/5 files <listed/in-dir>, 85 blocks used,
  100 unused, 100 in largest
```

M(ake

On the menu: M(ake

This command creates a directory entry with the specified file name.

M(ake requires you to enter a file name. Wild card characters are not allowed. The file size specification option is extremely helpful because it allows you to determine the size of the file you are creating. If you omit the size specification, the filer creates the file by consuming the largest unused area of the disk. The file size is determined by following the file name with the desired number of blocks, enclosed in square brackets ([ ]). The file size specification was described earlier under File and Volume Names.

Text files must be an even number of blocks with the smallest possible text file four blocks long (two for the header, and two for text). M(ake enforces these restrictions; if you try to M(ake a text file with an odd number of blocks, M(ake rounds the number down.

M(ake can be used to create a file (with no data) for future use, to extend the size of a file (using the size specification), or to recover a lost file.

EXAMPLE:

Prompt:

```
Make what file?
```

Response:

```
MYDISK:FILE.TEXT[28]
```

The preceding procedure creates the file FILE.TEXT on the volume MYDISK:. It is made to be 28 blocks long to occupy the first unused 28-block area on the volume.

M(ake is used to create .SVOL files which contain subsidiary volumes. For more information about this, see the paragraph "Subsidiary Volumes".

N(ew

On the menu: N(ew

This command clears the work file.

If you have a work file, the system displays this prompt:

```
Throw away current work file?
```

Pressing the Y key clears the work file, while pressing the N key returns you to the outer level of the filer.

If <work file name>.BACK exists, then the system displays the following prompt:

```
Remove <work file name>.BACK ?
```

Entering Y removes the file in question, while N leaves the .BACK file alone, but does create a new work file.

When N(ew is successful, the system displays this message:

```
Workfile cleared
```

O(n/off-line

On the menu: O(n/off-line

This command mounts or dismounts subsidiary volumes. With the filer menu displayed, press the O key. The system displays the following menu:

```
Subsidiary Volume: M(ount, D(ismount, C(lear
```

Press the M key. The system displays the following prompt:

```
Mount what vol ?
```

To dismount a subsidiary volume, press the D key. The system displays the following prompt:

```
Dismount what vol ?
```

To dismount all the subsidiary volumes, press the C key. The system immediately dismounts all the subsidiary volumes that are currently mounted.

Suppose that a principal volume, P__VOL:, contains the following files and that the prefix is set to P__VOL.

```
P__VOL:
   FILE1.TEXT
   FILE1.CODE
   VOL1.SVOL
   FILE2.TEXT
   FILE2.CODE
   DOC1.SVOL
   FUN.SVOL
```

To mount subsidiary volumes on P_VOL:, you can respond to the mount prompt with the file name of the .SVOL file as in the following examples.

```
Mount what vol ? VOL1.SVOL RETURN

Mount what vol  ? VOL1.SVOL,FUN.SVOL RETURN

Mount what vol ? P_VOL:=RETURN

Mount what vol ? #5:=RETURN
```

The first example mounts VOL1:; the second mounts VOL1: and FUN:; the third mounts all three subsidiary volumes on P_VOL:; and the fourth example mounts all subsidiary volumes on the disk in drive #5:.

To dismount any of these volumes, you can respond to the dismount prompt with the Volume ID as in the following examples.

```
Dismount what vol ? #14:

Dismount what vol ? VOL1: RETURN

Dismount what vol  ? VOL1:, DOC1:, FUN: RETURN
```

The first example dismounts the subsidiary volume associated with the device number #14. The second example dismounts VOL1:, and the third example dismounts three subsidiary volumes.

NOTE

When mounting a subsidiary volume, represent it as a file name (VOL1.SVOL). When dismounting a subsidiary volume, represent it as a volume name (VOL1:). For more information see "Subsidiary Volumes", Chapter 3.

P(refix

On the menu: P(refix

This command changes the current default volume to the volume that you specify.

This command requires you to enter a volume name or device number. The specified volume need not be on-line.

If you specify a device number (such as #5), then the new default prefix is the name of the volume in that device. If no volume is in the device when P(refix is used, the default prefix remains the device number (such as #5); thereafter, any volume in the default device is the default volume.

Since P(refix tells you the volume name of the new default volume, you may respond to its prompt with a colon (:) to determine the current default volume's name. To return the prefix to the booted or root volume, you may respond with an asterisk (*).

To use this command, select P(refix and the following prompt is displayed:

```
Prefix titles by what vol?
```

You should enter the desired volume name or device number.

CAUTION

When using only a device number for the prefix, *remember* that any disk in the device is the default disk. In this situation, it is very easy to assume that the system is prefixed to a particular disk, exchange the disks, and write over a valuable file or destroy information.

Q(uit

On the menu: Q(uit

This command terminates the filer and returns you to the command menu.

R(emove

On the menu: R(em

This command removes file entries from the directory.

R(emove requires one file specification for each file you wish to remove. Wild cards are legal. Size specification information is ignored.

EXAMPLE:

Given the example files (assuming that they are on the default volume):

```
EXAMPLE.TEXT
COPYIT.CODE
MEMO.TEXT
RUNIT.CODE
```

Prompt:

```
Remove what file?
```

Response:

```
RUNIT.CODE
```

Removes the file RUNIT.CODE from the volume directory.

NOTE

To remove SYSTEM.WRK.TEXT and/or SYSTEM.WRK.CODE, use the N(ew command, not R(emove. Using R(emove may confuse the system.

Before completing any R(emove commands, the filer displays the following prompt:

Prompt:

```
Update directory?
```

Pressing the Y key causes all specified files to be removed. Pressing the N key returns you to the outer level of the filer without removing any files.

As noted before, wild card characters in R(emove commands are legal.

EXAMPLE:

Prompt:

```
Remove what file?
```

Response:

```
=CODE
```

This causes the filer to remove RUNIT.CODE and COPYIT.CODE.

Pressing the wild card question mark (?) causes the R(emove command to display a prompt questioning the removal of each file on a volume. This is useful for cleaning out a directory and for removing a file that has (inadvertently) been created with a nonprinting or otherwise invalid character in its name.

CAUTION

Remember that the filer considers an equal sign (=) by itself to specify every file on the volume. Pressing an equal sign alone causes the filer to remove every file on the directory. (To escape from this situation, press N in response to the Update directory? prompt.)

S(ave

On the menu: S(ave

This command saves the work file under the file name you specify.

The entire file specification is not necessary. If the volume ID is not given, the default disk is assumed. Wild card characters are not allowed, and the size specification option is ignored.

EXAMPLE:

Prompt:

```
Save as what file?
```

Enter a file name of ten characters or less. This causes the filer to automatically remove any old file having the given name and to save the work file under that name. For example, pressing X in response to the prompt causes the work file to be saved on the default disk as X.TEXT. If a code file has been compiled since the last update of the work file, that code file is saved as X.CODE.

If a file already exists with the name given, the S(ave command responds with:

```
Destroy old <file name>?
```

Pressing the Y key causes the old file to be replaced; any other reply exits the S(ave command.

The filer automatically appends the suffixes .TEXT and .CODE to files of the appropriate type. If you enter AFILE.TEXT in response to the prompt, the filer saves the file as AFILE.TEXT.TEXT. The filer ignores any illegal characters in the file name except colon (:). If the file specification includes a volume ID, the filer assumes that you wish to save the work file on another volume.

For example, if in response to the filer prompt `Save as what file?` you enter VOL1:FILE1, the system then displays the following message.

```
MYDISK:SYSTEM.WRK.TEXT-->VOL1:FILE1.TEXT
```

T(ransfer

On the Menu: T(rans

This command copies the specified file or volume to the given destination.

T(ransfer requires you to enter two specifications: one for the source file or volume and another for the destination file or volume, separated by either a comma or RETURN. Wild card characters are permitted in file name specifications only, and size specification information is recognized for the destination file.

EXAMPLE:

Assume that you wish to transfer the file DOCU.TEXT from the disk MYDISK to the disk BACKUP.

Prompt:

```
Transfer what file ?
```

Response:

```
MYDISK:DOCU.TEXT
```

Prompt:

```
To where?
```

Response:

```
BACKUP:NAME.TEXT
```

NOTE

On a one-drive machine, do not remove the source disk until the system displays that prompt asking you to insert the destination disk.

Prompt:

```
Put in BACKUP: press <space> to continue
```

You should remove the source disk, insert the destination disk, and press the space bar.

In any case, when the T(ransfer is complete, the filer displays this message:

```
MYDISK:DOCU.TEXT-->BACKUP:NAME.TEXT
```

You may want to transfer a file without changing its name. The filer enables you to do this easily by allowing the character dollar sign ($) to replace the file name in the destination file specification. In the above example, had you wished to save the file DOCU.TEXT on BACKUP under the name DOCU.TEXT, you could have done so like this:

```
MYDISK:DOCU.TEXT,BACKUP:$
```

CAUTION

Avoid entering the second file specification with the file name completely omitted.

For example, if in response to the T(ransfer command prompt, `Transfer what file`, you respond by entering MYDISK:DOCU.TEXT,BACKUP:, the system will display the following prompt.

```
Destroy BACKUP: ?
```

Pressing Y causes the directory of BACKUP: to be destroyed.

NOTE

If the file to be transferred is two blocks long or less, the system will not display the warning prompt.

You may transfer files to volumes that are not storage volumes, such as CONSOLE: and PRINTER:, by specifying the appropriate volume ID in the destination file specification. Do not specify a file name for a communication device. The system will ignore it. Make sure the device is on-line before the transfer.

EXAMPLE:

Prompt:

```
Transfer what file?
```

Response:

```
DOCU.TEXT
```

Prompt:

```
To where?
```

Response:

```
PRINTER:
```

The preceding procedure causes DOCU.TEXT to be written to the printer.

You may also transfer from storage devices, provided they are input devices. The source file must end with an EOF (which is a soft character configurable using the SETUP utility); otherwise, the filer will not know when to stop transferring. File names accompanying a communication device are ignored.

Wild card characters are recognized in the T(ransfer command. If the source file specification contains a wild card character, and the destination file specification involves a storage device, then the destination file specification must also contain a wild card character.

The subset-specifying strings in the source file specification are replaced by the analogous strings in the destination file specification (replacement strings). Any of the subset-specifying or replacement strings may be empty. The filer considers the file specification denoted by an equal sign (=) to specify every file on the volume.

EXAMPLE:

The volume MYDISK contains the files:

```
PODA-1, PODB-1, PODC-1
```

The destination disk is SUCCESS.

Prompt:

```
Transfer what file?
```

Response:

```
P=1,SUCCESS:M=2
```

The system then displays the following listing:

```
MYDISK:PODA-1 --> SUCCESS:MODA-2
MYDISK:PODB-1 --> SUCCESS:MODB-2
MYDISK:PODC-1 --> SUCCESS:MODC-2
```

The filer will try to transfer every file on the disk if you specify the equal sign (=) as the source file name.

Using the equal sign (=) as the destination file name specification replaces the subset-specifying strings in the source specification with nothing. You may use the question mark (?) in place of the equal sign. Using the question mark, you will be asked to verify each operation before it is performed.

You may transfer a file from a volume to the same volume by specifying the same volume ID for both source and destination file specifications. This is frequently useful when you wish to relocate a file on the disk. Specifying the number of blocks desired causes the filer to copy the file in the first available area of at least that size. If you do not specify a size, the file is written in the largest unused area.

If you specify the same file name for both source and destination on a same-disk transfer, then the filer rewrites the file to the size-specified area and removes the older copy.

EXAMPLE:

Prompt:

```
Transfer what file?
```

Response:

```
#4:QUIZZES.TEXT,#4:QUIZZES.TEXT[20]
```

The preceding procedure causes the filer to rewrite QUIZZES.TEXT in the first 20-block area encountered (counting from block 0) and to remove the previous version of QUIZZES.TEXT.

You can also transfer an entire volume from one disk to another. The file specifications for both source and destination should consist of only volume ID; for example, Disk1:, Disk2:. Transferring a storage volume to another storage volume wipes out the destination volume so that it becomes an exact copy, including directory, of the source volume.

NOTE

> System disks contain an invisible bootstrap which is placed on them by the Disk Format Utility. This bootstrap is *not* copied when you do a disk-to-disk transfer.

EXAMPLE:

Assume that you want an extra copy of the disk MYDISK: and transfer to a disk called EXTRA:

Prompt:

```
Transfer what file?
```

Response:

```
MYDISK:,EXTRA:
```

Prompt:

```
Destroy EXTRA: ?
```

CAUTION

If you enter Y, the directory of EXTRA: will be destroyed, with EXTRA becoming an exact copy of MYDISK. An N response returns you to the outer level of the filer with no transfer taking place.

This volume-to-volume transfer process is a good backup procedure. Use the C(hange command to change the name of the backup disk. The two disks should not have the same name because this may confuse the system.

Although you can transfer a volume (disk) to another, using a single disk drive, it is tedious. This is because the transfer in main memory reads the information in rather small chunks, and a great deal of disk juggling is necessary to complete the transfer.

V(olumes

On the menu: V(ols

This command lists volumes currently on-line with their associated volume (device) numbers.

The following listing is a typical display.

```
Vols on-line:
  1    CONSOLE:
  2    SYSTERM:
  4 #  WNCHSTR: [10404]
  5 #  FLOPPY1: [  640]
  6    PRINTER:
  9 #  FLOPPY2: [  640]
Root vol is - WNCHSTR:
Prefix is   - FLOPPY2:
```

Root vol is the system disk or boot disk. Prefix is indicates the default disk. Storage volumes are indicated by #.

After each disk volume, the number of 512-byte blocks that it contains is given in square brackets. This can be useful if the system uses disks of varying storage capacities. In the preceding example, the Winchester disk on-line in drive #4: contains 10,404 blocks of storage capacity, and the flexible disks on-line in drives #5: and #9: each contain 640 blocks.

The V(olumes command also displays the mounted subsidiary volumes. The name of the principal volume and the name of the starting block are given for each subsidiary volume listed.

The following listing is an example.

```
Vols on-line:
  1    CONSOLE:
  2    SYSTERM:
  4 #  WNCHSTR: [10404]
  5 #  FLOPPY1: [  320]
  6    PRINTER:
  9 #  FLOPPY2: [  640]
 13 #  DOCS:    [ 3000] on volume WNCHSTR:
                starting at block 400
 14 #  PROGRMS: [ 3000] on volume WNCHSTR:
                starting at block 3700
 15 #  FUN:     [ 3000] on volume WNCHSTR:
                starting at block 7040
Root vol is - WNCHSTR:
Prefix is   - FLOPPY2:
```

In this example, three subsidiary volumes on WNCHSTR: are mounted. They use device numbers #13:, #14:, and #15:. Each of these volumes contains 3,000 blocks.

W(hat

On the menu: W(hat

This command identifies the name and state (saved or not) of the work file.

EXAMPLE:

```
Work file is DOC1:STUFF (not saved)
```

X(amine

On the menu: X(amine

This command attempts to physically recover suspected bad blocks.

You must specify the name of a volume that is on-line.

EXAMPLE:

Prompt:

```
Examine blocks on what vol?
```

Response:

```
<volume ID>
```

Next Prompt:

```
Block-range ?
```

You should have just performed a bad block scan and should enter the block number(s) returned by that scan. If any files are endangered, the following prompt should appear:

Prompt:

```
file(s) endangered:
        <file name>
         Fix them?
```

Pressing the Y key causes the filer to examine the blocks and return either of the messages:

```
Block <block-number> may be ok
Block <block-number> is bad
```

In which case the bad block has probably been fixed, or block <block-number> is bad. If block <block-number> is bad, the filer offers you the option of identifying the block(s) as BAD. Blocks marked BAD are not moved during a K(runch and are rendered unavailable and effectively harmless (though they do reduce the amount of room on the disk).

An N response to the `Fix them?` prompt returns you to the outer level of the filer.

CAUTION

A block that is fixed may contain garbage. The message *May be ok* should be translated to mean probably physically ok. Fixing a block means that the block is read, is written back out to the block, and is read again. If the two reads are the same, the message is *May be ok*. If the reads are different, the block is declared bad and may be marked as such if so desired.

Z(ero

On the menu: Z(ero

This command initializes the directory on the specified volume, rendering the previous directory irretrievable.

EXAMPLE:

Prompt:

```
Zero dir of what vol ?
```

Response:

```
<volume ID>
```

Prompt:

```
Destroy <volume name> ?
```

Pressing the Y key brings this prompt:

```
Duplicate dir ?
```

If you press the Y key, a duplicate directory is maintained. This is advisable because if the disk directory is destroyed, a utility program called COPYDUPDIR can use the duplicate directory to restore the disk.

The next prompt appears only if there was a directory on the disk before the Z(ero command was used:

Prompt:

```
Are there 640 blks on the disk ? (y/n)
```

Press the Y key to accept that number of blocks and skip the next prompt. Press the N key to receive this prompt:

```
# of blocks on the disk ?
```

Enter the number of blocks desired. This number varies depending upon your particular disks.

The next prompt is:

```
New vol name ?
```

Enter any valid volume name.

Prompt:

```
<new volume name> correct ?
```

Press the Y key to accept the name. Press the N key to return to the prompt which requests a new volume name. If the filer succeeds in writing the new directory on the disk, this message is displayed:

```
<new volume name> zeroed
```

4

System Editor

INTRODUCTION

The UCSD p-System on the Texas Instruments Professional Computer uses the Screen-Oriented Editor. This editor allows you to create, alter, and read text files. Text files contain human-readable material such as memos or manuscripts.

SCREEN-ORIENTED EDITOR

In order to use the editor, SYSTEM.EDITOR must reside on a disk which is on-line. Also, the file SYSTEM.MISCINFO must be configured for your terminal. If this has not already been done for you, configure it with the SETUP utility described in Chapter 5.

The Window into the File

The Screen-Oriented Editor is specifically for use with video display terminals (or cathode ray tubes, CRTs), most of which have 24-line screens. The editor usually uses the first line of the display unit to display its menu. Therefore, most of the time it displays 23 lines of text within the file. Using the editor, you may view any part of the file in 23-line segments.

You actually look into the file through a window that the editor provides. Although you can access the whole file by using editor commands, you can view only a portion of it through the window in the display unit. When an editor command takes you to a position in the file that is not presently displayed, the window moves to show you that new portion of the file.

The Cursor

The cursor is a small rectangular box that is logically located between the character to its left and the character space on which it rests. You position the cursor to indicate to the editor how its commands are to affect the text. For example, the editor will insert text to the left of the character space on which the cursor rests.

You can move the cursor to any specific location in a file; at that point, it then represents your exact position in the file. The window shows the portion of the file that surrounds the cursor; to see another portion of the file, move the cursor. The cursor follows the commands of the editor. For example, if you delete portions of the file, you move the cursor to indicate the beginning and extent of the deletion.

In this chapter, all text examples are shown in uppercase, with the cursor denoted by an underline or a lowercase character.

The Menu

The editor displays a menu at the top of the display unit to remind you of the current command and the options available for that command. The most commonly used options appear in the menu. The following is an example of the editor's first-level menu, called the edit menu.

```
>Edit: A(djust C(opy D(el F(ind I(nsert J(ump
   K(ol M(argin P(age ? [version]
```

Notation Conventions

The notation used in this chapter corresponds to the notation the editor uses to prompt you. The system uses angle brackets (< >) to indicate a single key like the **RETURN** key (<return>) or the space bar (<space>).

Enter FILE NAME <return> means to enter the name of the file and then press the **RETURN** key. You may use either lowercase or uppercase when entering editor commands.

Editing Environment Options

The editor has two chief modes of operation: one for entering and modifying programs and another for entering and modifying English (or any other language) text. The first mode includes automatic indentation; the second includes automatic text filling. For more information on these two options, see the description of the E(nvironment option of the S(et command.

Command Hierarchy

The command menu is the first or highest level of the command hierarchy. To enter the system editor, press the E key from the command menu. The system will display the edit menu:

```
>Edit: A(djust C(opy D(el F(ind I(nsert J(ump
   K(ol M(argin P(age ? [version]
```

The edit menu is the second level of the command hierarchy, as is the filer menu and all the other menus that you can display from the command menu.

For example, call the I(nsert option, press the I key. The system now displays the third level of the command hierarchy:

```
>Insert: Text {<bs> a char, <del> a line}
   [<ext> accepts, <esc> escapes]
```

Repeat Factors

The F(ind and R(eplace commands, as well as most of the cursor-movement keys, allow repeat factors. A repeat factor allows you to specify the number of times a command should be performed by the editor. For example, press the 2 key and then press the R key to select the R(eplace command. The editor will display this third-level menu:

```
>Replace[2]: L(it V(fy <targ><sub> =>
```

The number 2 that you entered appears inside the square brackets to indicate that the system will perform the specified function two times.

If you do not specify a repeat factor, the default (assumed) factor is 1. Use a slash (/) to specify that a function should be performed as many times as possible.

Direction Indicator

The direction indicator determines whether the cursor will be moved in the forward direction or in the reverse direction. For example, if the direction indicator is forward, the cursor moves to the right (toward the end of the file) when you press the space bar. If the direction indicator is reversed, then the cursor moves left (toward the beginning of the file) when you press the space bar.

The first character in the menu indicates the global direction. A right angle bracket (>) indicates movement to the right, and a left angle bracket (<) indicates movement to the left. To change the global direction, press the left or right angle brackets on the keyboard. When you enter the editor, the global direction is right.

Moving the Cursor

The special keys described in this section enable you to move the cursor in a number of ways. Global direction affects the space bar, **RETURN** key, and the **TAB** key. It does not affect the arrow keys and the **BACKSPACE** key.

Pressing the equal sign (=) moves the cursor to the beginning of the last text that was most recently inserted, found, or replaced. The equal sign works from anywhere in the file and is not affected by the global direction. An I(nsert, F(ind, or R(eplace saves the position (within the work file) of the beginning of the insertion, find, or replacement.

Pressing the = moves the cursor to that position and saves the cursor location. If you perform a C(opy or a D(elete between the beginning of the file and that absolute position, the cursor will not jump to the start of the insertion, because that absolute position has then been lost.

The J(ump command moves the cursor to the beginning or end of a file, or to a previously defined marker anywhere within the file (see the S(et M(arker command). The P(age command moves the display unit window forward (or backward) by one screen and positions the cursor to the beginning of the line. These commands are described in "Screen-Oriented Editor" in this chapter.

The following list summarizes the keys which move the cursor.

Not affected by current global direction:

↓ (down-arrow)	Moves the cursor down
↑ (up-arrow)	Moves the cursor up
→ (right-arrow)	Moves the cursor right
← (left-arrow)	Moves the cursor left
BACKSPACE	Moves the cursor left

Motion determined by global direction:

space bar	Moves the cursor one space in the global direction
TAB	Moves the cursor to the next tab stop
RETURN	Moves the cursor to the beginning of the next line

These keys change the global direction to backward:

<	Left angle bracket
,	Comma
–	Minus sign

These keys change the global direction to forward:

>	Right angle bracket
.	Period
+	Plus sign

You can use repeat factors with any of the cursor movement keys listed above.

You cannot move the cursor outside the text of the program. For example, after the N in BEGIN in the following example, press the → key; this moves the cursor to the W in WRITE. Similarly, at the W in WRITE-('TOO WISE ');, use the ← key to back up after the N in BEGIN.

```
BEGIN_
  WRITE('TOO WISE ');

BEGIN
  _WRITE('TOO WISE ');
```

In the following example, if you must change the WRITE('TOO WISE '); found in the third line to a WRITE('TOO SMART ');, you must first move the cursor to the correct position.

For example, if the cursor is at the P in PROGRAM STRING1;, go down two lines by pressing the ↓ key twice. To mark the positions the cursor occupies, labels a, b, and c are used in the following example. The a marks the initial position of the cursor; the b marks the cursor position after the first ↓ key; and the c marks the cursor after the second ↓ key.

```
aROGRAM STRING1
bEGIN
cWRITE('TOO WISE ');
```

Now, using the → key, move the cursor until it sits on the W of WISE. Note that with the use of the ↓ key, the cursor appears to be outside the text (c). In this case, the editor considers the cursor to be at the W in WRITE; when you press the first ← key, the cursor jumps to the R in WRITE. However, when the cursor is displayed outside the text, it is actually on the closest character to the right or left.

F(ind and R(eplace

Both F(ind and R(eplace operate on delimited strings. The editor has two string storage variables. One, called <targ> by the menus, is the target string and is used by both commands; while the other, called <sub> by the R(eplace menu, is the substitute string and is used only by R(eplace.

Enter these strings when using F(ind or R(eplace. Once entered, they are saved by the editor and may be reused.

When you enter a string, you must use a special character to delimit (mark) the beginning and end of the string. For example, /fun/, $work$, and "gismet" represent the strings fun, work, and gismet, respectively. The editor allows any character that is not a letter or a number to be used as a delimiter.

F(ind and R(eplace operate in either of two search modes: literal and token. These modes are stored by the S(et E(nvironment command and can be changed by it, or they may be temporarily overridden using the F(ind or R(eplace commands.

In the literal mode, the editor looks for any occurrences of the target string. In the token mode, the editor looks for isolated occurrences of the target string. The editor considers a string isolated if it is surrounded by spaces or other punctuation. For example, in the sentence Put the book in the bookcase., using the target string book, the literal mode finds two occurrences of book, while the token mode finds only one: the word book isolated by spaces.

In addition, the token mode ignores spaces within strings, so that <space> comma <space> (,) and comma (,) are considered the same string.

When using either F(ind or R(eplace, you may use the strings previously entered by pressing the S key. For example, entering RS/<any-string>/ causes the R(eplace command to search for an occurrence of the previous target string and replace it with <any-string>. R/<any-string>/S causes the next occurrence of <any-string> to be replaced with the previous substitute string.

To find out the current contents of the <targ> and <sub> strings are in, use the S(et E(nvironment command.

Work Files

When you enter the editor, the system reads and displays the work file. If you have not already created a work file, the editor will display the following prompt:

```
>Edit: No work file is present.
  File? ( <ret> for no file )
```

There are three ways to respond to this prompt:

1. With a name, for example STRING1<ret>. The file named STRING1.TEXT is now retrieved. The file STRING1 could contain a program, also called STRING1, as in the following example. After typing the name, a copy of the text of the first part of the file appears on the display unit.

```
PROGRAM STRING1;
BEGIN
  WRITE('TOO WISE');
  WRITE('YOU ARE');
  WRITELN(',');
  WRITELN('TOO WISE');
  WRITELN('YOU BE')
END.
```

2. With a RETURN key. This response indicates that you wish to start a new file. The only thing visible on the display unit after this response is the Edit menu. Press the I key to begin inserting a program or text.

3. With the ESC key. This response stops the editor, causing the system to display the command menu.

Using I(nsert

To use the I(nsert option, press the I key from the Edit menu. Place the cursor on top of the letter before which you want to make an insertion. The cursor must be in the correct position before pressing the I key. From the point of insertion, the rest of the line is moved toward the right side of the display unit. If the insertion is long, that part of the line is moved down to allow room on the display unit.

After pressing the I key, the system displays the following prompt:

```
>Insert: text {<bs> a char,<del> a line} [<etx>
  accepts, <esc> escapes]
```

NOTE

ETX is a generic p-System key that is activated by pressing and holding the CTRL key and then pressing the C key (CTRL-C).

Suppose the cursor is at the W in WISE (see the example in "Work Files"). Enter SMART. The word appears on the display unit as it is entered in the following example.

```
BEGIN WRITE('TOO SMART__      WISE ');

BEGIN WRITE('TOO SMARTWISE ');
```

While in I(nsert, you can insert a carriage return by pressing the RETURN key. The editor then starts a new line. Notice that a carriage return starts a new line with the same indentation as the previous one. This is often convenient when entering program text.

Using D(elete

D(elete works like I(nsert. Move the cursor to the W in WISE (see the preceding example) and press the D key to select the D(elete command. The system then displays the following prompt:

```
>Delete: < > <Moving commands> {<etx> to delete,
  <esc> to abort}
```

Press the space bar four times. Each time you press it, a letter disappears from the display unit. Pressing the BACKSPACE key causes a character to reappear. By pressing the CTRL-C keys the proposed deletion is carried out; or pressing the ESC key causes the proposed deletion to reappear and remain part of the text.

To delete a carriage return at the end of a line, call D(elete and then press the space bar until the cursor moves to the beginning of the next line.

Leaving the Editor

When all text changes and additions have been made, press the Q key to leave the editor. The system then displays the following menu.

```
>Quit:
U(pdate the workfile and leave
E(xit without updating
R(eturn to the editor without updating
W(rite to a file name and return
```

Using the U(pdate option saves a copy of the file on disk as SYSTEM.WRK.TEXT. This file is your work file.

The W(rite option saves the file under whatever name you wish. The file is not necessarily your work file.

R(eturn simply returns you to the editor without saving anything on disk.

E(xit leaves the editor without saving anything. Any changes or additions to the file are lost permanently.

SCREEN-ORIENTED EDITOR COMMANDS

The screen-oriented editor commands are covered in alphabetical order in this section.

A(djust

On the menu: A(djust

Repeat factors are allowed in conjunction with the arrow keys within A(djust.

Press the A key from the edit menu. This displays the following menu:

```
>Adjust: L(just R(just C(enter <arrow keys>
  {<etx> to leave}
```

The A(djust option moves a line to the left or to the right. The → and ← keys move the line on which the cursor is located. Each time you press a → key, the whole line moves one space to the right. The ← key moves the line one space to the left.

To adjust more than one line, use the ↑ or ↓ keys; the line above or below the previously adjusted line is automatically adjusted by the same amount.

Pressing the L key justifies the line to the left margin, pressing the R key justifies it to the right margin, and pressing the C key centers the line between the margins. Press the ↑ and the ↓ keys to duplicate the adjustment on preceding (succeeding) lines.

Use the S(et E(nvironment command to alter the margins.

The system repositions the cursor to the beginning of the last line adjusted. Press the CTRL-C keys to exit the A(djust command; the ESC key will not work here.

C(opy

On the menu: C(opy

Repeat factors are not allowed.

Press the C key from the edit menu. The following menu is displayed.

```
>C(opy: B(uffer F(rom file <esc>
```

To copy text into the copy buffer, press the D key to use the D(elete command. Perform the delete, but press the ESC key so that the text is not deleted but only copied into the buffer. The C(opy command allows text to be copied into the current text from one of two sources: a temporary buffer called the copy buffer, and a text file on disk. To copy from the copy buffer, press the B key. The editor immediately copies the contents of the buffer into the file, starting at the location of the cursor when you pressed C. The buffer may be recopied until you change the contents of the buffer.

When the C(opy function ends, the cursor goes to the end of the copied text.

The following commands affect the copy buffer.

1. D(elete: When you press the CTRL-C keys, the buffer is loaded with the deletion. When you press the ESC key, the buffer is loaded with what would have been deleted.

2. I(nsert: When you press the CTRL-C keys, the buffer is loaded with the insertion. When you press the ESC key, the copy buffer is emptied.

3. Z(ap: If you use the Z(ap command, the buffer is loaded with the deletion.

4. M(argin: This command causes the copy buffer to be left empty.

To copy text from another file, press the F key. The system then displays the following menu.

```
>C(opy: From what file[marker,marker]?
```

Any file may be specified; .TEXT is assumed. The markers are optional and are used for copying part of a file.

To copy part of a file, you must have previously S(et markers at the beginning and end of the text you wish to copy. You may use two markers, or the file's beginning or end as a marker. For example, if you specify [,marker] or [marker,], the file is copied from the start of the file to the marker or from the marker to the end of the file.

D(elete

On the menu: D(el

Repeat factors are not allowed.

To select the D(elete option, press the D key from the edit menu. The following prompt is displayed:

```
>Delete: < > <Moving commands> {<etx> to delete,
  <esc> to abort}
```

First, place the cursor where you want to delete text. The D(elete option uses an *anchor* at this initial position. As you move the cursor away from the anchor, characters disappear. Moving back toward the anchor restores those characters to the text file. To accept the deletion, press the **CTRL-C** keys; to escape, press the **ESC** key.

Within the D(elete option, all cursor-moving actions are valid, including repeat factors and global direction.

The following procedure shows how to use the D(elete option with reference to this example.

```
PROGRAM STRING2;
BEGIN
  WRITE('TOO WISE ');
  WRITELN('TO BE.')
END.
```

1. Move the cursor to the E in END.

2. Press < (this changes the direction to backward).

3. Press the D key.

4. Press the RETURN key twice. After pressing the RETURN key once, the cursor moves to the position in front of the W in WRITELN, and WRITELN('TO BE.'); disappears. After the second return, the cursor appears before the W in WRITE with that line gone.

5. Now press the CTRL-C keys. After deletion, the program appears as shown in the following example.

```
PROGRAM STRING2:
BEGIN
END.
```

The two deleted lines have been stored in the copy buffer, and the cursor has returned to the anchor position. If you wish, you may now use C(opy to copy the two deleted lines to any other place in the file.

Whenever a deletion is larger than the available copy buffer space, the editor will display the following warning.

```
There is no room to copy the deletion. Do you wish
  to delete anyway? (y/n)
```

Pressing the Y or y key indicates a yes answer; any other character escapes the D(elete option.

F(ind

On the menu: F(ind

Repeat factors are allowed.

To use the F(ind option, press F from the edit menu. The system will display one of the following prompts (depending upon how T(oken definition is set in S(et E(nvironment):

```
>Find[n]: L(it <target>  =>
>Find[n]: T(ok <target>  =>
```

(Where n is the repeat factor given before pressing the F key; this number is one if you gave no repeat factor.)

The F(ind option locates the nth occurrence of the <target> string, starting from the cursor position and moving in the global direction (shown by the arrow at the beginning of the menu). The cursor stops at the position immediately after this occurrence.

To search in the token or the literal mode, press the appropriate character (either L or T, respectively), before entering the target string.

If the string does not occur within the text file between the cursor and the end or beginning of the file (depending on global direction), the system displays the following message:

```
ERROR: Pattern not in the file
  Please press <spacebar> to continue.
```

The following paragraphs show how to use the F(ind option.

```
PROGRAM STRING1;
BEGIN
  WRITE('TOO WISE ');
  WRITE('YOU ARE');
  WRITELN(',');
  WRITELN('TOO WISE ');
  WRITELN('YOU BE.')
  END.
```

In the STRING1 program (the preceding example), with the cursor at the first P in PROGRAM STRING1, press the F key. When the prompt appears, enter WRITE. Single quote marks must be entered. The prompt with the user response is shown in the following listing.

```
>Find[1]: L(it <target>  =>'WRITE'
```

The cursor jumps immediately to the character following the E in the first WRITE.

In the STRING1 program with the cursor on the E in END., enter <3F. This entry finds the third occurrence of the pattern in the reverse direction. When the menu appears, enter /WRITELN/. The menu with the user response is shown in the following listing.

```
<Find[3]: L(it <target>  =>/WRITELN/
```

The cursor will move to a position immediately after the N in WRITELN.

On the first find, enter F/WRITE/. This locates the first WRITE. Now enter FS. The cursor appears after the second WRITE.

I(nsert

On the menu: I(nsert

Repeat factors are not allowed.

To call the I(nsert option, press the I key from the edit menu. The system then displays the following menu:

```
>Insert: Text {<bs> a char,<del> a line} [<etx>
  accepts, <esc> escapes]
```

Characters are entered into the text file as they are pressed, starting from the position of the cursor. This includes the carriage return character. Nonprinting characters are echoed with the nonprinting character symbol (usually a question mark (?); this can be changed by using SETUP). To make corrections while still in I(nsert, use the **BACKSPACE** key to remove one character at a time or press and hold the **SHIFT** key and press the **DEL** key to remove an entire line. Backspacing past the beginning of the insertion causes the system to display an error message.

Create the text file with the I(nsert command, using the modes selected with the S(et E(nvironment commands. Use S(et E(nvironment for selecting the A(uto-indent and the F(illing options.

If A(uto-indent is selected, pressing the **RETURN** key causes the cursor to start the next line with an indentation equal to the indentation of the line above it. If A(uto-indent is false, pressing the **RETURN** key returns the cursor to the first position of the next line.

If F(illing is selected, the editor forces all insertions to be between the right and left margins. It does this by automatically inserting returns between words whenever the right margin would have been exceeded and by indenting to the left margin whenever a new line is started. The editor considers anything to be a word that is between two spaces or between a space and a hyphen.

Pressing the **RETURN** key twice in succession creates a new paragraph. In other words, a paragraph is either a block of text delimited by blank lines or command lines (see S(et), or the beginning or end of a text file. The first line of a paragraph may be indented differently than the remaining text (see S(et E(nvironment).

If both A(uto-indent and F(illing are selected, A(uto-indent controls the left-margin, while F(illing controls the right-margin. You may change the level of indentation by pressing the space bar and **BACKSPACE** key immediately after pressing the **RETURN** key.

EXAMPLE 1:

With A(uto-indent true, the following sequence creates the indentation shown in the following example.

```
ONE<RETURN>
<space><space>TWO<RETURN>
THREE<RETURN>
<BACKSPACE>FOUR
```

```
ONE       original indentation
  TWO     indentation changed by <space><space>
  THREE   <RETURN> causes auto-indentation to level of line above
 FOUR     <BACKSPACE> changes indentation from level of line above
```

EXAMPLE 2:

With F(illing selected (and A(uto-indent not selected) the following sequence creates the indentation shown in the following example.

```
ONCE UPON A TIME
THERE- WERE
```

```
ONCE UPON A          Auto-returned when next word would exceed margin
TIME THERE-          Auto-returned at hyphen
WERE-
```

You can force the cursor to the left margin of the display unit by pressing the **CTRL-Q** keys (ASCII DC1) twice.

F(illing also causes the editor to adjust the margins on the portion of the paragraph following the insertion. This adjustment does not affect any line beginning with the command character (see S(et), and such a line terminates a paragraph.

You may readjust a filled paragraph by using the M(argin command, but only if F(illing is TRUE and A(utoindent is FALSE. This may be very useful if you wish to change the margins of a document (which may be done with S(et E(nvironment).

The global direction does not affect I(nsert, but is indicated by the direction of the arrow on the menu.

If an insertion is made and accepted, that insertion is available for use in C(opy. However, if the ESC key is pressed, there is no string available for C(opy.

J(ump

On the menu: J(ump

Repeat factors are not allowed.

Upon entering J(ump, the following menu appears:

```
>JUMP: B(eginning E(nd M(arker <esc>
```

Pressing the B (or E) key moves the cursor to the beginning (or the end) of the file. Pressing the M key displays the following prompt:

```
Jump to what marker?
```

Markers are user-defined names for positions in the text file. See the M(arkers option of the S(et command for more information.

K(olumn

On the menu: K(ol

Repeat factors are not allowed.

K(olumn displays the following menu:

```
>Kolumn: <vector keys> {<etx>, <esc> CURRENT line}
```

You may move all of a line which lies to the right of the cursor to the left by using the ← or to the right by using the →. Using the ↑ or ↓ applies the same column adjustment to the line above or below. Press **CTRL-C** to leave K(olumn. You can use **ESC**, but it only rejects the changes made most recently to the current line.

NOTE

When using K(olumn, each time you press the ← key one character is deleted at the cursor. Characters deleted are not saved in the copy buffer as in D(elete. It is easy to do this, so be careful when using K(olumn.

M(argin

On the menu: M(argin

Repeat factors are not allowed.

M(argin realigns the paragraph (where the cursor is located) to fit within the current margins. All of the lines within the paragraph are justified to the left margin, except the first line, which is justified to the paragraph margin. You can set all these global margins with the S(et E(nvironment command.

The cursor may be located anywhere within the paragraph when you press the M key.

The following examples show margin settings and examples of paragraphs that use those settings.

Left-margin, 0
Right-margin, 40
Paragraph-margin, 8

```
        This quarter, the equipment is
different, the course materials are
substantially different, and the course
organization is different from previous
quarters.  You will be misled if you
depend upon a friend who took the course
previously to orient you to the course.
```

Left-margin, 8
Right-margin, 40
Paragraph-margin, 0

```
This quarter, the equipment is
        different, the course materials
        are substantially different, and
        the course organization is
        different from previous quarters.
        You will be misled if you depend
        upon a friend who took the course
        previously to orient you to the
        course.
```

A paragraph is any block of text delimited by blank lines, lines beginning with a command character, or the beginning or end of the text file. If the text file or the paragraph is especially long, the system may remain blank for several seconds while M(argin completes its work. When M(argin finishes, the system redisplays the paragraph. M(argin never splits a word; it breaks lines at spaces or at hyphens.

M(argin will not affect a line if the line starts with a command character. The command character must be the first nonblank character in the line. M(argin treats lines beginning with the command character as blank lines. The command character itself is any character so designated using the S(et E(nvironment command.

P(age

On the menu: P(age

Repeat factors are allowed.

Moves the cursor one screen height in the global direction. If a repeat factor is used, several screen heights are traversed. The cursor remains on the same line on the display unit, but is moved to the start of the line.

Q(uit

On the menu: Q(uit

Repeat factors are not allowed.

Q(uit displays the following menu:

```
>Quit:
U(pdate the workfile and leave
E(xit without updating
R(eturn to the editor without updating
W(rite to a file name and return
```

Select one of the four options by pressing U, E, R, or W All other characters are ignored.

U(pdate:

Stores the file just modified as SYSTEM.WRK.TEXT; then leaves the editor. SYSTEM.WRK.TEXT is the text portion of the work file.

E(xit:

This leaves the editor immediately. Any modifications made since entering the editor are not recorded on disk. All editing during the session is irrecoverably lost, unless you have already used the W(rite option of Q(uit to save the work.

R(eturn:

Returns to the editor without updating. The cursor is returned to the exact place in the file it occupied when Q was pressed. This command is frequently used after unintentionally pressing Q. It is also useful when you wish to make a backup to your file in the middle of a session with the editor.

W(rite:

This option calls up a further menu:

```
>Quit:
Name of output file (<cr> to return)-->
```

The file may now be given any proper name. If it is written to the name of an existing file, the new copy replaces the old file. Use $ to write to the same name that the file had when you entered the editor. Alternatively, you can abort, Q(uit, at this point by pressing the **RETURN** key instead of entering a file name; you will return to the editor. If the file is written to disk, the editor displays the following:

```
>Quit
Writing.....
Your file is 1978 bytes long.
Do you want to E(xit from or R(eturn to the
  editor?
```

R(eplace

On the menu: R(plc

Repeat factors are allowed.

Upon entering R(eplace, one of the two menus in the following example appears, depending on the global mode. In this example, a repeat factor of four is assumed.

```
>Replace[4]:L(it V(fy <targ><sub> =>

>Replace[4]:T(ok V(fy <targ><sub> =>
```

R(eplace finds the target string (<targ>) exactly as F(ind would, and replaces it with the substitution string (<sub>).

The verify option (V(fy) allows you to examine each <targ> string found in the text so that you can decide if it is to be replaced. To use this option, press the V key before pressing the target string.

The following menu appears whenever R(eplace has found the <targ> pattern in the file and verification has been requested:

```
>Replace: <esc> aborts, 'R' replaces, ' ' doesn't
```

Pressing the R key at this point causes the replacement to take place, and the next target to be sought. Pressing the space bar causes the next occurrence of the target to be sought. At any point, pressing the **ESC** key aborts the R(eplace.

With V(erify, this process continues until the repeat factor is exhausted or until the target string can no longer be found.

With R(eplace, if the target string cannot be found, the following menu appears.

```
ERROR: Pattern not in the file. Please press
  <spacebar> to continue.
```

R(eplace places the cursor after the last string that was replaced.

EXAMPLE 1:

Enter RL/Low//High/ like this:

```
>Replace[1]: L(it V(fy <targ> <sub> =>L/Low//High/
```

This command will change: Lowly to Highly.

Literal is necessary because the string Low is not a token, but part of the token Lowly.

EXAMPLE 2:

In the Token mode, R(eplace ignores spaces between tokens when finding patterns to replace. This example concerns the following two lines.

```
WRITE(',');
WRITE( ',');
```

Press the 2 key and then the R key from the Edit menu. The system then displays the following menu:

```
>Replace[2]: L(it V(fy <targ> <sub>
```

Enter /(',')/.LN. Immediately after entering the last period, the following two lines replace the previously listed lines:

```
WRITELN;
WRITELN;
```

S(et

On the menu: S(et

Repeat factors are not allowed.

Upon entering S(et, the following menu appears:

```
>Set: E(nvironment M(arker <esc>
```

S(et M(arker

When editing, it is particularly convenient to be able to jump directly to certain places in a long file by using markers set in the desired places. Once a marker is set, you can jump to it by using the M(arker option in J(ump.

Move the cursor to the desired marker position, enter S(et, and press M for M(arker. The following prompt appears:

```
Set what marker?
```

You may give markers names of up to eight characters and then press the RETURN key. Marker names are case-sensitive, that is, the lowercase and uppercase of the same letter are considered to be different characters. The marker is entered at the position of the cursor in the text. If you use the name of a marker that already exists, it will be repositioned.

Twenty markers are allowed in a file at any one time. You will receive the following display if you try to set more than 20 markers:

```
Marker ovflw.  Which one to replace?
  (Type in the letter or <sp>)
a) name1   b) name2   c) name3   d) name4
e) name5   f) name6   g) name7   h) name8
i) name9   j) name10  k) name11  l) name12
m) name13  n) name14  o) name15  p) name16
q) name17  r) name18  s) name19  t) name20
```

Choose a letter in the range of a through t; that space will now be available for use in setting the desired marker.

S(et E(nvironment

You can set the editing environment to a mode that is very convenient for word processing or more structured kinds of editing (such as programming text or special tables). When in S(et, press the E key (for E(nvironment). The following display will then appear:

```
Environment:  {options} <spacebar> to leave
   A(uto indent    True
   F(illing        False
   L(eft margin    1
   R(ight margin   80
   P(ara margin    6
   C(ommand ch     ^
   S(et tabstops
   T(oken def      True

   3152 bytes used, 29612 available.

Editing:  SCHEDULE TEXT
Created March 10, 1982; last updated
  March 24, 1982 (revision 10)
Editor Version [IV 1 f4].
```

The line that begins Editing: identifies the file currently being edited. If the file has just been created but not named, the line reads:

```
Editing:  unnamed
```

By pressing the appropriate letter, you may change any or all of the options.

A(uto Indent: — A(uto-indent affects only insertions. Refer to the section on I(nsert. A(uto-indent is set to true (turned on) by pressing the A and T keys and to false (turned off) by pressing the A and F keys.

F(illing: — F(illing affects I(nsert and M(argin. (Refer to those sections.) F(illing is set to true (turned on) by pressing the F and T keys and to false by pressing the F key twice.

When F(illing is true, the margins set in E(nvironment are the margins that affect I(nsert and M(argin. They also affect the C(enter and justifying commands in A(djust. To set a margin, press the L, R, or P keys, followed by a positive integer and a space. The positive integer entered replaces the previous value. Margin values must be four digits or less.

C(ommand Ch: — The command character (C(ommand ch:) affects the M(argin command and the Filling option in I(nsert. (Refer to those sections.) Change the command character by pressing C, followed by any character. For example, entering C* changes the command character to *. This change is reflected in the menu. The command character was principally designed as a convenience for using text formatting programs whose commands are indicated by a special character at the beginning of a line.

S(et Tabstops: — The editor allows you to set tab stops. From the E(dit command menu, press S(et E(nvironment and then press S(et tabstops. The system will display the following interface menu:

```
Set tabs: <right, left vectors> C(ol#
  T(oggle tab <etx>

T----T----T----T----T----T----T----T----T----T----T

  Column #1
```

The cursor will start at position one in the line of Ts and dashes (-). The line Column # 1 indicates the position of the cursor. To set or remove a tab, move the cursor to the desired location, using the right or left vector keys, or press the C key and enter the desired column number. Press the T key to insert a tab or delete a tab.

Pressing the T key changes the indicator from a dash to T, pressing the T key again in the same column changes the T back to a dash. The system displays the current column number of the current cursor position and updates it each time you press a right/left vector key or C(olumn command.

T(oken Def: — This option affects F(ind and R(eplace. Set Token to be true by entering TT and to false by entering TF. If Token is true, Token is the default; if Token is false, Literal is the default.

V(erify

On the menu: V(erify

Repeat factors are allowed.

The current window is redisplayed, and the cursor is repositioned at the center of the text on the screen.

X(change

On the menu: X(ch

Repeat factors are not allowed.

Upon entering X(change, the following menu appears:

```
eXchange: TEXT <vector keys> {<etx>, <esc>
  CURRENT line}
```

Starting from the position, X(change replaces characters in the file with characters you enter.

For example, in the file in the first example below, with the cursor at the W in WISE, entering XSM replaces the W with the S and then the I with the M. This leaves the line, as shown in the second example below, with the cursor before the second S.

```
WRITE('TOO wISE ');
```

```
WRITE('TOO SMsE ');
```

Press the CTRL-C keys to accept the actions of X(change. Press the ESC key to leave the command with no changes recorded only in the last line altered.

The X(change command ignores the global direction; exchanges are always forward.

You may use the arrow keys, **BACKSPACE**, **RETURN**, and **TAB** keys to move the cursor. X(changes move forward from wherever the cursor is moved to.

While in X(change, the terminal's KEY TO INSERT CHARACTER inserts one space at the cursor's location, and the KEY TO DELETE CHARACTER deletes a single character at the cursor's location. These keys may be specified with SETUP (see Chapter 5). Initially, they are **INS** and **DEL**, respectively.

Z(ap

On the menu: Z(ap

Repeat factors are allowed.

Z(ap deletes all text between the start of what was previously found, replaced, or inserted and the current position of the cursor. Use this command immediately after a F(ind, R(eplace, or I(nsert. If more than 80 characters are being zapped, the editor asks for verification.

The position of the cursor after the previous F(ind, R(eplace, or I(nsert is called the equal mark. Pressing = places the cursor there.

Whatever you deleted by using the Z(ap command is available for use with C(opy, unless there is not enough room in the copy buffer. If this is the case, the editor then asks if you want to Z(ap anyway.

Z(ap is not allowed after certain commands that might scramble the buffer. These commands are: A(djust, D(elete, K(olumn, and M(argin.

5

Utility Commands

INTRODUCTION

This chapter covers several utility programs that will help you use the p-System on the Texas Instruments Professional Computer. The utility programs are code files that you X(ecute to provide such services as:

- Printing text files
- Recovering lost files
- Configuring the p-System for your particular keyboard and terminal
- Helping to analyze and debug programs that you write
- Showing you the internal details of files

The first utility described here, called PRINT, falls into the first category. The next three utilities fit in the second category. They are called COPYDUPDIR, MARKDUPDIR, and RECOVER. The utility called SETUP belongs to the third category. The remaining p-System utilities described here help you to configure the p-System on the Texas Instruments Professional Computer and to format disks.

THE PRINT UTILITY

The PRINT utility provides a simple way for p-System users to print text files. The Screen-Oriented Editor in the p-System makes it easy to create and manipulate text (including documents, memos and programs). The PRINT utility makes it just as easy to produce printed versions of such text. PRINT can break a document into pages and put headings on each, including the page number. In addition, there are a variety of options for controlling the line spacing and vertical margins of the printed version.

PRINT complements the other two principal mechanisms within the p-System for printing text files (the T(ransfer operation in the Filer and the Print Spooler). The big advantage of using the Print Spooler is that printing can go on in parallel with other operations, such as text editing. This can be a big time saver. PRINT can be used with the Spooler because PRINT's output can be sent to a disk file. The Spooler can then be used to print that formatted file.

PRINT has been designed to work with a wide variety of printers. It makes minimal assumptions about special printer control features, and it can be used with either continuous forms or manual single sheet loading.

The following describes the simplest uses of PRINT. You may never need to know more. If you do, read the rest of this description, which provides a systematic description of all of PRINT's facilities.

Simple Uses of PRINT

To invoke PRINT, simply X(ecute it from the command menu of the p-System. PRINT immediately shows a menu of the available commands. Some of these cause immediate action by the program (such as printing a document); others allow you to set up configuration parameters that will guide a subsequent printing operation (such as what disk file to print).

Most of these configuration parameters are initially set up by PRINT for the most common printing situations. In particular, we assume:

- That you are using continuous paper in your printer (rather than single sheets)
- That each page can hold at least 66 lines of printing (or 11-inch paper with 6 lines per inch)
- That your printer advances the paper to a new page when the p-System sends an ASCII form feed control character to it

If these built-in choices meet your needs, using PRINT is very simple and consists of four steps (once you have PRINT running):

- Enter the name of the file to be printed, using the I(nput option on the menu.
- Use G(o to start the printing. After a file is printed, use I(nput to select another file and G(o again to print it.
- If you need to cause a page advance on the printer to tear off the printout(s) you have made, A(dvance should do the trick.
- Finally, when you are done with a printing session, use Q(uit to leave PRINT.

If the printouts produced by this process are not what you would like, or if some of the assumptions above do not apply in your situation, read the rest of this description to discover how PRINT can be configured to serve your needs better. For instance, you can use PRINT with a printer that requires each sheet to be individually loaded.

Interacting with PRINT

Just as in the rest of the p-System, you interact with PRINT by making choices from a menu of options. There are four kinds of commands. They may:

- Cause immediate actions. A(dvance, for instance, moves the paper in the printer to the next page.

- Prompt you to enter a sequence of characters and then press the RETURN key. In the case of I(nput, these characters are a file name.

- Request that you enter an integer number. This number must be positive and have four digits or fewer. This style of interaction is used when you choose the initial page number for your heading lines.

- Give you a Yes or No choice. Respond by pressing Y or N. This style of response is used with the D(ouble space option, for instance.

When the PRINT menu is displayed, you can enter ? to see a description of the PRINT functions and commands.

Other than the principal menu of commands, which occupies most of your display unit, PRINT does most of its communication with you through the top line of the screen. Once you have selected a command, prompts appear on this line to direct you. Error messages are also shown on this line and are usually left there until you press the space bar to indicate that you have noticed the message.

Controlling the Layout of Pages

PRINT allows you to specify the P(age length you are using and the sizes of the T(op and B(ottom margins that you desire. All these are specified in units of print lines. At the top of a page, after T(op margin lines of empty space, a heading line is printed (which may have the date and page number, for instance). A blank line follows the heading. Here is a diagram of a page, with these parameters shown:

```
Top of page
      .
      .
T(op margin
 blank lines
      .
      .
Header line
Blank line
First text line
      .
      .
      .
Text
      .
      .
      .
Last text line
      .
      .
B(ottom margin
 blank lines
      .
      .
Bottom of page
```

PRINT does not attempt to control the horizontal placement of the text it processes. Lines are transferred to the printer exactly as they appear in the file being printed.

The standard header line contains a page number, the name of the file being printed, and the current date (as maintained by the p-System). The format of this line can be changed, as described in the next section. Here is an example header line in the standard format:

```
Page 3.  File is "MYVOL:MYFILE". Printed on
  January 3, 1983.
```

The initial page number for a file is ordinarily 1. If you want your page numbers to start differently, use P(age number before printing your file.

The D(ouble space and N(umbered lines options can be used to control those aspects of your printout's appearance.

The Content of Pages

As mentioned above, the normal operation of PRINT is to transfer lines without change from the file being printed to the printer.

There are two exceptions to this general principle. First, if a line has a special flag character as its first item, it may be a COMMAND line that gives directions to PRINT, but is not intended to be printed itself. The two characters after the flag are examined to see if they correspond to a valid PRINT command. If they do, the command is accepted by PRINT, and the line is not transferred to the printer. Otherwise, the line is printed as usual.

The second exception is that a line may contain the ESCAPE SEQUENCE flag. This character can be anywhere in the line. As with the commands, it is the characters after the escape sequence flag which determine what happens. In general, however, the escape sequence is replaced by other text (for instance, the current page number).

These two flag characters can be changed either from the PRINT menu (using the E(scape and C(ommand options) or by command lines embedded in a file being printed.

Only the first two characters after the command flag are significant. Additional characters are ignored. (Therefore, .INCLUDE and .INCLEMENT are both treated as include commands.) Commands may be in either upper or lowercase.

Commands may have parameters. The first parameter must be separated from the command name by one or more blank characters. All parameters must be enclosed in quotes. Either single quotes (') or double quotes (") may be used, but both ends of the parameter must be marked by the same character (that is, "myfile" or 'myfile').

The commands available in PRINT are as follows:

INCLUDE

This command has one parameter, which is a file name. Printing of the current file is temporarily suspended and the included file is processed. When the end of the included file is reached, processing resumes on the principal file. The included file cannot itself contain any INCLUDE commands. Page numbering is continuous across included files.

INCLUDE allows a large document to be spread among several p-System files, but still be printed with a single PRINT operation. For example:

```
.INCLUDE 'MYVOL:MYFILE'
```

PAGE

This command has no parameters. Its effect is to cause an immediate page advance on the printer.

This command is useful when the page breaks that are automatically inserted by PRINT are not the page breaks that you want. For example:

```
.PAGE
```

HEADER

This command has one parameter, which becomes the new specification of the header line which is printed at the top of each page. The header line can also be changed from the PRINT menu.

You can use this command in a document file to establish a page heading for the printed version that is specific to the document. For example:

```
.HEADER "My document-Page \page"
```

COMMAND

This command has one parameter, a single character (which still needs to be enclosed in quotes). The character becomes the new command line flag character.

This infrequently used command allows you to choose the character that introduces command lines. For example:

```
.COMMAND '^'
```

ESCAPE

This command is similar to COMMAND, except that the one-character parameter becomes the new escape sequence flag.

Just as with the command flag, you may want to change the escape sequence flag if the standard one conflicts with something in your text file. For example:

```
.ESCAPE '!'
```

All characters are significant in escape sequences. There are three standard ones, which are translated as follows when found in a line about to be printed:

- PAGE: the escape sequence is replaced by the current page number.
- FILE: the input file name (either the main file, or the include file, whichever is active).
- DATE: the escape sequence is replaced by the current p-System date in the form "January 1, 1983."

A principal application of these escape sequences is in the header line printed at the top of each page. You can change the format of that line either in the PRINT menu or with the .HEADER command line in the file being printed. For example, the header

```
Memorandum of Understanding(\date)-Page \page
```

would produce printed heading lines like the following:

```
Memorandum of Understanding (January 13, 1983)-
  Page 43

Memorandum of Understanding (May 18, 1983)-Page 7
```

The provision for changing the header line within the file means that you can have different headers on different pages. It would be easy, for instance, to have a blank header on the first page and some specific header on subsequent pages.

Output Methods

PRINT directs its output by default to the device PRINTER:. You can easily change this definition, however, from the PRINT menu. You could, for instance, set the output file to be a disk file or a serial communications port. The disk file possibility can be quite useful, since it allows you to store the paginated output of PRINT for later transfer to a printer. If the Print Spooler is used for that transfer, you can take advantage of the Spooler's ability to overlap printing with other p-System operations, particularly text editing.

PRINT is intended to work with printers which use continuous forms, but also with printers that must be loaded with each individual sheet of paper. The S(top before each page option in the PRINT menu controls which kind of printer is assumed. If the single-sheet variety is selected, you are prompted to load the printer before each page is printed.

On many single-sheet-oriented printers, the paper must be inserted about an inch past the printing mechanism so that pinch rollers can guide it. If you are using such a printer, you may want to reduce the P(age size and possibly change the T(op margin, as well. For instance, if your printer prints six lines per inch and you are using standard 11-inch paper, you might reduce the P(age size from 66 lines to 60 lines.

Most printers can interpret the ASCII form-feed character and advance the paper to the next page. If your printer cannot, turn off the U(se form feed option, and the form feed character will be replaced by the printing of a series of empty lines. The effect will be the same as a form feed, as long as PRINT's page size and margin options are properly set.

PRINT Execution Shortcuts

If the standard settings of the PRINT options suit your needs most of the time, the use of PRINT is simple and convenient. If, however, you generally need to change one or more of the options to do your printing, PRINT could be more awkward to use. The M(ake script command has been included to address this potential need.

This command produces a script file that will change the options from their defaults on entry to PRINT to the values that exist at the time that M(ake script is invoked. You can also include in this script a command that invokes itself, thus reducing your keystrokes even further.

When you select M(ake script, you are first asked to name the script file you want produced. If you want this to be a .TEXT file, you must include the suffix in the title you supply. The advantage of a .TEXT file is that it can be easily read or modified by a p-System editor. A disadvantage is that it is at least four blocks long, whereas a typical nontext file script is only one block long.

The next prompt asks you to enter the name by which PRINT should be invoked. Your response is basically used as the response to an X(ecute prompt, so whatever you would use there is appropriate. If you provide an empty response to this prompt (that is, an immediate press of the **RETURN**, the program invocation step is left out of the generated script altogether.

After this second prompt, PRINT produces the script.

Here is an example of M(ake script, along with a subsequent invocation of PRINT.

```
Enter name of script file: MYPRINT

Enter name for invoking print: *PRINT

Execute what file? i=MYPRINT
```

In the first line above, the script file is dubbed MYPRINT (with no suffix). The second line indicates that the PRINT program is to be found on the system disk, with the indicated file name. The third line is the invocation of PRINT via the newly created script. The script will execute the program and set all the options as they existed at the time the script was made.

If the second response above had been empty, then an equivalent X(ecute string would have been *PRINT i=MYPRINT.

Summary of Menu Items

By selecting any of the options below, you can:

I(nput	Choose the file to be printed.
O(utput	Choose the destination of the print operation.
G(o	Print the input file on the output, according to the current option settings.
A(dvance	Skip to the next page on the output.
M(ake script	Build a script file which will invoke PRINT with the current option settings.

Q(uit	Leave PRINT.
D(ouble space	Select single- or double-spaced output.
N(umber	Cause the lines on each page to be numbered.
S(top	Specify whether single sheet loading or continuous forms are assumed by PRINT.
U(se ASCII FF	Specify whether the form feed character or a sequence of empty lines is used to separate output pages.
F(irst page	Specify the page number on the first page of a document.
T(op margin	Specify the number of blank lines between the top of the page and the header line.
B(ottom margin	Specify the number of blank lines between the last line of text and the bottom of the page.
P(age size	Specify the number of lines per page.
E(scape	Choose the character which starts an escape sequence.
C(ommand	Choose the character which starts a command line.
H(eader	Specify the contents of the heading line at the top of each printed page.

Summary of Command Lines

By using one of the following commands, you can:

INCLUDE	Effectively insert an additional file into the document being printed in place of the I(nclude command.
PAGE	Cause an immediate page break.
HEADER	Specify the contents of the heading for subsequent pages.
COMMAND	Change the command line flag character.
ESCAPE	Change the escape sequence flag character.

Summary of Escape Sequences

When any of the following escape sequences occur, the indicated text is substituted:

PAGE	The current page number.
FILE	The current input file name.
DATE	The current calendar date as maintained by the p-System.

THE COPYDUPDIR UTILITY

COPYDUPDIR copies the duplicate directory of a disk into the primary directory location. In certain situations, a duplicate directory may help rescue directory information that is garbled or lost.

The Z(ero command of the filer can create a duplicate directory, as can the MARKDUPDIR utility. Once a duplicate directory has been created, the filer maintains it along with the primary directory.

To use this utility, X(ecute COPYDUPDIR. The system then displays a prompt asking for the drive in which the copy is to take place. If the disk does not currently contain a duplicate directory, COPYDUPDIR displays a prompt stating so. If the duplicate directory is found, then COPYDUPDIR displays a prompt asking if you want to destroy the directory in blocks 2 through 5. Press the Y key to execute the copy; any other character aborts the program.

THE MARKDUPDIR UTILITY

MARKDUPDIR creates a duplicate directory on a disk that does not currently contain one.

Be sure that blocks 6 through 9 are free for use. If they are not, use T(ransfer or a backward K(runch to free them. To determine if these blocks are available, do an extended listing in the filer and check to see where the first file starts. If the first file or unused area starts at block 6, then the disk does not have a duplicate directory. However, if the first file or unused area starts at block 10, then the disk already has a duplicate directory.

EXAMPLE:

```
SYSTEM.PASCAL     106    1-Jan-83     10     Codefile
  .
  .
  .

OR

<unused>            4                   6
SYSTEM.PASCAL     106    1-Jan-83     10     Codefile
  .
  .
  .
```

Both of the preceding cases indicate disks that have no duplicated directory. The following listing is a directory of a properly marked disk:

```
SYSTEM.PASCAL    106   1-Jan-83    10    Codefile
  .
  .
  .
```

To mark a directory, execute MARKDUPDIR. The system will display a prompt asking which drive contains the disk to be marked (#4 or #5). MARKDUPDIR checks to see if blocks 6 through 9 are free. If they are not, the system displays a prompt asking if you are sure they are free. Press the Y key to continue; any other character will abort the program. Be sure that the space is free before marking it as a duplicate directory; otherwise, you will lose file information.

THE RECOVER UTILITY

The RECOVER utility attempts to recreate the directory of a disk whose directory has accidentally been destroyed.

The RECOVER utility displays several yes/no prompts. These must be answered with uppercase letters; lowercase letters are ignored.

The following paragraphs comprise a list of RECOVER's prompts followed by a description of appropriate responses to those prompts.

Prompts:

```
Recover [IV.1 H]
USER'S DISK IN DRIVE#(0 exits):_
```

Enter the number of the drive that contains the disk to be RECOVERed (for example: 4). Press the **RETURN** key.

Prompt:

```
USER'S VOLUME ID:
```

Enter a volume name, which is recorded on the disk. The name should be in uppercase letters.

Prompt:

```
HOW MANY BLOCKS ON DISK?
```

The system displays this prompt if the number of blocks recorded in the (damaged) directory is not a valid number. The response depends on the types of disks you have.

RECOVER reads each entry in the disk's directory and checks it for validity. Entries with errors are removed. Valid entries are saved, and RECOVER displays `ENTRY.NAME found` (or something similar).

When all the directory entries have been checked, saved, or discarded, RECOVER displays the following prompt:

```
Are there still IMPORTANT files missing? (Y/N)
```

If you press the N key, RECOVER displays the following prompt:

```
GO AHEAD AND UPDATE DIRECTORY? (Y/N)
```

If you press the N key, RECOVER finishes executing without doing anything.

If you press the Y key, RECOVER saves the reconstructed directory and displays the following prompt:

```
WRITE OK
```

Then RECOVER terminates.

If you press the Y key in response to the `Are there still IMPORTANT files missing?` prompt, RECOVER searches those areas of the disk still not accounted for by the (partially) reconstructed directory. Text files and code files are detected, and appropriate directory entries are created for them. If RECOVER cannot determine the original name of a text file it has found, it creates a directory entry for DUMMY# #.TEXT or DUMMY# #.CODE (where the # # are two unique digits). If a code file has a PROGRAM name, it is given that name. If this would create a duplicate entry in the directory, digits are used; for example, RECOVER first restores SEARCH.CODE and then SEARCH00.CODE.

RECOVER cannot detect data files, since their format is not system-defined. To recover data files, you must use the PATCH utility, described in the *UCSD p-System Program Development Reference Manual.*

If RECOVER restores a text file with an odd number of blocks, this probably means that the end of the text file was lost. Use the editor to make sure this is the case.

You should recover code files if linking was necessary.

When RECOVER has finished its pass over the entire disk, it displays the following prompt:

```
GO AHEAD AND UPDATE DIRECTORY? (Y/N)
```

THE RAM CONFIGURATION UTILITY

The Texas Instruments Professional Computer allows you to have from 64K to 256K bytes of main memory. (One K is 1024 bytes.) The p-System always uses the first 64K (for its internal stack and heap). If you *only* have 64K, then executable code also has to reside there. (Executable code includes the major p-System components as well as programs that you run.) However, if you have more than 64K, you may decide to have executable code reside in the extra memory. This gives your programs and the p-System components more room to run. It also gives the internal workings of the p-System more room, since the entire 64K is available just for that.

This concept is called the *external code pool.* Main memory in excess of 64K can either be used for the external code pool, or for RAM disk, or for some combination of these two.

A RAM disk is like a physical disk except that it resides in main memory. It may contain files. These files must be read into the lower 64K of main memory or the external code pool in order to be used. (This is just like reading files in from a physical disk.)

For this reason, you may decide to make the external code pool as large as possible. However, it can only grow to 64K. A RAM disk can be as large as you wish (within the physical restraints of your memory).

The RAM configuration (CONFIG.RAM.CODE) utility allows you to define the system parameters for RAM disk and for extended memory.

The extended memory option allows either 0, 32K, 48K, or 64K bytes of additional memory to be allocated to the external code pool. The parameters defining the extended code pool are stored in the file SYSTEM.MISCINFO. These parameters become effective once the system is booted from a disk containing the new SYSTEM.MISCINFO. Memory not allocated to the extended code pool is utilized by the RAM disk automatically once this utility is run.

To run this utility, X(ecute PSYS:CONFIG.RAM. This prompt is displayed:

```
Enter the unit number of the system disk (4,5,9,10,11)? _
                        unit 0 exits
```

Press the number of a drive that contains a system disk that you want to configure (or press the 0 key to quit the utility). The system disk must contain the file SYSTEM.INTERP. If it does not, an error is displayed and you can enter another unit number or press the 0 key to quit.

If a valid system disk is located in the specified unit, the RAM configuration utility presents the following information:

```
RAM Configurator: I(nstructions, C(onfigure, Q(uit? _
```

For instructions, press the I key; if you want to modify the extended code pool size, press the C key. Otherwise, press the Q key and the utility returns to the system menu.

If you select the configure option, the utility prompts for the new extended code pool size:

```
Ram configuration on volume VOLNAME:

    Extended code pool size:
```

The extended code pool size must be either 0, 32, 48, or 64. This is the number of kilobytes (1 kilobyte = 1024 bytes) to be allocated to the extended code pool. If there is enough memory, the RAM disk base follows the extended code pool immediately. The new parameters take effect the next time the system is booted from that disk.

The following table describes the possible RAM disk and extended memory configurations:

Total Size of RAM (K Bytes)	Extended Code Pool Size (K Bytes)	RAM Disk Size in Blocks
64	0	0
128	0	128
	32	64
	48	32
	64	0
192	0	256
	32	192
	48	160
	64	128
256	0	384
	32	320
	48	288
	64	256

THE REMOTE CONFIGURATION UTILITY

This utility provides a mechanism for initializing the communications parameters of the Texas Instruments Professional Computer's remote ports to values other than the defaults. The new port parameters only become effective once the system has been rebooted from the disk where the parameters are stored.

To execute the remote configurator, use the X(ecute command of the system menu to execute the CONFIG.REM code file. The remote configurator presents the following prompt:

```
Enter the unit number of the system disk (4,5,9,10,11)? _
  unit 0 exits
```

Enter the unit number which contains the system disk with the file SYSTEM.INTERP. Press the 0 key to quit the utility. The remote configurator displays the current information for the remote port and prompts for modifications as follows:

```
Remote Configurator: U(nit, A(ctive port, B(usy sensing,
 P(arity, S(top bits, C(haracter length, R(ate, T(imeout,
 D(efault, Q(uit, ESC(exit

                    U(nit                   Remote
                    A(ctive port:           Serial port 1
                    B(usy sensing:          None
                    R(ate:                  300
                    P(arity:                None
                    S(top bits:             1
                    C(haracter length:      7
                    T(imeout:               0
```

Enter the first letter of the parameter name to change the parameter value. For example, press the T key to change the T(imeout value. The optional values are then displayed for the parameter selected.

The following table defines the default values and optional settings for each of the remote port control parameters.

Parameter	Default Value	Optional Values
Unit	Remote	Printer
Active port	Serial port 1	Serial port 2
Busy	None	Xmit On/Off, Reverse channel ON, Reverse channel OFF
Baud rate	300	75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 9600
Parity	Odd	Even, None
Stop bits	1	2
Character length	7	8
Timeout	0	0-32767 (0.1 second intervals)

U(nit allows you to select whether the parameters apply to the remote port or a serial printer port. Use Q(uit to write the parameters to the system disk selected. The parameters take effect next time the system is booted from that disk. Use D(efault to select the default values (shown in the preceding table) for the B(usy, baud R(ate, P(arity, S(top bits, C(haracter length, and T(imeout parameters. Press the ESC key to exit the utility without making any changes.

The A(ctive Port parameter defines which serial port is being configured. The B(usy parameter controls busy sensing: choose Transmit On/Off (also called DC1 ready and DC3 busy), Reverse channel ON, Reverse channel OFF, or NO busy sensing. For the baud R(ate parameter, enter the letter displayed beside the desired baud rate. For the P(arity parameter choose Odd, Even, or None parity checking. The S(top bits value can be 1 or 2. The C(haracter length can be 7 or 8 (bits). Timeout values are in 0.1 second intervals (60 = 6 seconds).

NOTE

> Remember that the parameters for a serial printer port must match those on the printer itself. (You do not need to configure a port for a parallel printer.) Also, if the remote and printer units both refer to the same port, the printer parameters will be used for both units.

THE PRINTER CONFIGURATION UTILITY

This utility allows you to enable or disable the compressed print feature on your printer (if your printer is so equipped). You can also define which port (parallel or serial) is to be used for the printer if you have the optional serial port(s). The source for this utility is provided so that support for additional printers can be added.

To run the Printer Configuration utility, execute CONFIG.PTR. The utility first prompts for the unit number of the system disk as follows:

```
Enter the unit number of the system disk (4,5,9,10,11)? _
  unit 0 exits
```

Enter the number of a unit that contains a system disk with the files SYSTEM.INTERP and SYSTEM.MISCINFO (or press the 0 key to quit the utility). When a valid system disk is located in the specified unit, the following prompt is displayed:

```
C(ompressed print, S(pooler, P(ort, Q(uit _
```

Press the C key to enable or disable compressed printing.

NOTE

The printer must be connected and online to set this parameter.

The prompts are as follows:

```
Compressed print? (Y/N) _
```

Press the Y key (for Yes) to select compressed print. (On the TI 850 and TI 810 printers this is 16.7 characters per inch.) Press the N key (for No) to select standard print (10 characters per inch). The next prompt is for the model of your printer:

```
Printer model:
  A - TI 850
  B - TI 810
```

Press either the A or B keys to select the appropriate model. The source for this utility is provided so that other printers can be added to this list.

When the utility's prompt is displayed, press the S key to enable or disable print spooling. The following prompt will be displayed:

```
Spooler on? (Y/N) _
```

Press the Y key (for Yes) to enable print spooling, or press the N key (for No) to disable print spooling.

When the utility's prompt is displayed, enter P to select which printer port to use. The prompt is as follows:

```
Use P(arallel or S(erial port? _
```

Press the P key to select the parallel port, or the S key to select the serial port. The CONFIG.REM utility sets which serial port is used by the printer if more than one is installed.

When you have set your printer configuration and the prompt is displayed, press the Q key to quit the utility.

Changes to the S(pooler or P(ort parameters take effect only after you reboot the system using the disk that was configured. Since the bootstrap procedure always reads SYSTEM.INTERP from a disk (even if there is already a copy in RAM), it is usually necessary to configure a disk (other than the RAM disk) to change the S(pooler or P(ort parameters. It is not necessary to reboot to change the C(ompressed print parameter, but the printer must be connected and online when you make that change. If your system is configured to be loaded into the RAM disk, you must do one of the following before a new S(pooler parameter takes effect:

- Reboot by turning the power off, then on.
- Copy a new SYSTEM.MISCINFO file from a disk to the RAM disk and reboot using the Halt command.

THE DISK FORMATTING UTILITY

This utility provides a mechanism for formatting and zeroing new disks for use with the Texas Instruments Professional Computer. If the disk to be formatted is to be used as a system disk, be sure that BOOT.CODE is on the current system disk. (If BOOT.CODE is not found, you are asked if you want to continue.) X(ecute FORMAT on the PSYS disk. The following prompts are displayed:

```
Enter unit of disk to be formatted (4, 5, 9, 10): _
                        unit 0 exits
Place disk in unit xx, and press return...
```

(The xx in the second prompt reflects the drive number that you select in response to the first prompt.)

If you are formatting a previously initialized and zeroed volume, the utility asks you to verify that you wish to proceed as follows:

```
Destroy VOLNAME (Y/N)? _
```

Press the Y key to continue with the disk format or the N key to return to the system menu. If you press the Y key, the following prompts are displayed:

```
New vol name? _
NEWNAME: correct? _
```

If the name displayed is not correct, press the N key and enter the new volume name again. Otherwise, press the Y key. Once the formatting is complete, the number of blocks formatted is displayed. This utility writes the p-System boot loader onto the boot blocks of the disk (if BOOT.CODE is present) and initializes the volume directory.

CAUTION

Be sure to run the filer's X(amine command after formatting a disk. X(amine marks any bad blocks on the disk so those blocks will not be used for data storage.

THE SET DATE AND TIME UTILITY

This utility provides a mechanism for initializing the correct date and time. You should X(ecute SETTIME on the PSYS disk. The following prompts are displayed:

```
Enter the unit number of the system disk
  (4, 5, 9, 10, or 11)? _
  unit 0 exits
```

Enter the unit number on which you want to write the current date or time.

```
Set Date and Time: D(ate, T(ime, Q(uit_
  Current date is MM-DD-YY
  Current time is HH:MM:SS
```

Select the option desired and enter the new date or time in the format shown. (Enter hours as 0 through 23). The date established using the D(ate option of the F(iler is replaced by the date set by this utility.

SETUP

SETUP is provided as a system utility that *sets up* the p-System to properly interface with your hardware. It resides in a file called SETUP.CODE and changes a data file. This data file contains detailed information about your terminal and a few miscellaneous details about the system. You can run SETUP and change the data as many times as you want. After running SETUP, you must reboot so that the system starts using the new information. (In some cases, you can just I(nitialize.) You should also backup the old data file—at least until you are sure that the new one works.

SETUP takes its initial information from a file called SYSTEM.MISCINFO and can create a new version of that file called NEW.MISCINFO. The old version must be removed or renamed and the new version renamed SYSTEM.MISCINFO before some of the changed values it may contain can become effective.

SYSTEM.MISCINFO contains three types of information:

- Miscellaneous data about the system
- General information about the terminal
- Specific information about the terminal control keys

Running SETUP

Run SETUP like any other program that uses X(ecute. It will display the word INITIALIZING followed by a string of dots, and then the menu:

```
SETUP: C(HANGE T(EACH H(ELP Q(UIT [version]
```

To select any command, just type its initial letter.

When H(ELP appears on a menu, it can describe all the options on that menu.

T(EACH gives a detailed description of how to use SETUP. Most of it concerns input formats, which are mainly self-explanatory. However, if this is your first time running SETUP, you should look through all of T(EACH.

C(HANGE gives you the option of going through a prompted menu of all the items or of changing one data item at a time. In either case, the current values are displayed, and you have the option of changing them. If this is your first time running SETUP, the values given are the system defaults. You will find that your particular terminal probably requires different specifications.

Q(UIT has the following options:

- H(ELP)
- M(EMORY) UPDATE, which places the new values in main memory
- D(ISK) UPDATE, which creates NEW.MISCINFO on your disk for future use
- R(ETURN), which lets you go back into SETUP and make more changes
- E(XIT) which ends the program and returns you to the command menu

Please note that if you have a NEW.MISCINFO already on your disk, D(ISK) UPDATE will write over it.

When you use SETUP to change your character set, do not underestimate the importance of using keys you can easily remember and of making dangerous keys, like **BREAK** and **ESC**, hard to hit.

Once you have run SETUP, always backup SYSTEM.MISCINFO under another name. (OLD.MISCINFO is one suggestion.) Then, change the name of NEW.MISCINFO to SYSTEM.MISCINFO and reboot. You can also update to memory, alone, and continue using the system without rebooting. However, the results of your doing this may not always be what you wanted—and you will not have a backup. In general, M(emory update is a Q(uit option you will use only when experimenting. If you do run into trouble, remember that you can save the current in-memory SYSTEM.MISCINFO by running SETUP and performing a D(isk update *before* you change any data items.

When you reboot or I(nitialize, the new SYSTEM.MISCINFO will be read into main memory and the system will use its data, provided it has been stored under that name on the system disk (the disk from which you boot).

SYSTEM.MISCINFO — Data Items

The information in this paragraph is very specific; you may skip it on first reading. However, if you have a question about a certain data item, look in this paragraph. The items are ordered according to SETUP's menu.

Please note that SETUP frequently distinguishes between a character that is a key on the keyboard and a character that is sent to the display unit from the system.

There are a few characters you cannot change with SETUP. These are **RETURN** (<return>), **LINE FEED** (<lf>), ASCII DLE (**CTRL-P**), and **TAB** (**CTRL-I**). It is assumed that <return>, <lf>, and **TAB** are consistent on all terminals. ASCII DLE (data link escape) is used as a blank compression character. When sent to an output text file, it is always followed by a byte containing the number of blanks which the output device must insert. If you try to use **CTRL-P** for any other function, you will run into trouble.

BACKSPACE

When sent to the display unit, the backspace character should move the cursor one space to the left. Default: ASCII BS.

CODE POOL BASE

The CODE POOL BASE[FIRST WORD] and the CODE POOL BASE[SECOND WORD] are used to determine where the code pool resides on machines that use extended memory.

On the Texas Instruments Professional Computer, these two words, taken together, make up the 32-bit address for the base of the external code pool. The FIRST WORD is the most-significant 16 bits, and the SECOND WORD is the least-significant 16 bits. The default is:

FIRST WORD = 0
SECOND WORD = 0

CODE POOL SIZE

If the code pool is external, this entry indicates the number of WORDS, minus one, available for it to fill. This value may be as great as 32,767 (a 64K area). It may also be smaller, if desired, but it should be at least 12,287 (a 24K area). The base address of this area is given by the two code pool base words. This value is ignored if you are not using extended memory.

EDITOR ACCEPT KEY

This key is used by the Screen-Oriented Editor. When pressed, it ends the action of a command and accepts whatever actions were taken.

EDITOR ESCAPE KEY

This key is used by the Screen-Oriented Editor. It is the opposite of the EDITOR ACCEPT KEY—when pressed, it ends the action of a command and ignores whatever actions were taken.

EDITOR EXCHANGE-DELETE KEY

This key is also used by the Screen-Oriented Editor. It operates only while doing an X(change and deletes a single character.

EDITOR EXCHANGE-INSERT KEY

Like the EDITOR EXCHANGE-DELETE KEY, this only operates while doing an X(change in the Screen-Oriented Editor: it inserts a single space.

ERASE LINE

When sent to the display unit, this character erases all the characters on the line that the cursor is on.

ERASE SCREEN

When sent to the display unit, this character erases the entire display.

ERASE TO END OF LINE

When sent to the display unit, this character erases all characters, starting at the current cursor position to the end of the same line.

ERASE TO END OF SCREEN

When sent to the display unit, this character erases all characters, starting at the current cursor position to the end of the display.

FIRST SUBSIDIARY VOL NUMBER

This entry is the first unit number to be used as a subsidiary volume. For example, if you set it to 14, the first subsidiary volume is device #14:

NOTE

In previous versions of the UCSD p-System, only 6 storage volumes were allowed: 4, 5, and 9 through 12. Now the number of storage volumes is configurable. The devices from 9 through the unit number designated by the phrase FIRST SUBSIDIARY VOL NUMBER are now principal volumes. Subsidiary volumes start with the device number indicated by the phrase FIRST SUBSIDIARY VOL NUMBER. The number of subsidiary volumes is designated by MAX NUMBER OF SUBSIDIARY VOLS. The highest device number allowed for subsidiary volumes, standard block devices, or user-defined serial volumes (described below) is 127.

CAUTION

The FIRST SUBSIDIARY VOL NUMBER must be greater than 8 to allow space for all of the standard system units.

HAS 8510A

Should always be false.

HAS BYTE FLIPPED MACHINE

This should be TRUE.

HAS CLOCK

This character may be TRUE or FALSE. If your hardware has a line frequency (60 Hz) clock module, such as the DEC KW11, setting this bit TRUE allows the system to optimize disk directory updates. It also allows you to use the TIME intrinsic. If your hardware does not have a clock, this *must* be FALSE.

HAS EXTENDED MEMORY

When extended memory is not used, the code pool resides between the stack and the heap. If the code pool is removed from that memory space and placed in a different area altogether, then set HAS EXTENDED MEMORY to TRUE; otherwise, set it to FALSE. (An example of extended memory is a 128K-byte machine where the stack and heap reside within one 64K area, and the code pool resides within the other 64K area.)

HAS LOWER CASE

This may be TRUE or FALSE. It should be TRUE if you do have lowercase and want to use it. If you seem stuck in uppercase, even if this bit is TRUE, remember there is a soft alpha-lock: see KEY TO ALPHA LOCK.

HAS RANDOM CURSOR ADDRESSING

This character may be TRUE or FALSE.

HAS SLOW TERMINAL

This character may be TRUE or FALSE. When this bit is TRUE, the system's menus and prompts are abbreviated. You should leave this set to FALSE, unless your terminal runs at 600 baud or slower.

HAS SPOOLING

Set this to TRUE if the PRINT SPOOLER is to be used. If this field is true in SYSTEM.MISCINFO and SPOOLOPS has not been LIBRARYed into SYSTEM.PASCAL, the p-System will *not boot.*

HAS WORD ORIENTED MACHINE

May be TRUE or FALSE. If your processor uses byte addresses for memory references, this should be FALSE.

KEYBOARD INPUT MASK

Characters that are recieved from the keyboard will be logically ANDed with this value. For the typical ASCII keyboard, set this value to 7F hex (which throws away the eighth bit). For some keybords, which generate eight bit characters, use FF hex.

KEY FOR BREAK

When this key is pressed while a program is running, the program terminates immediately with a run-time error. Recommendation: use a key that is difficult to hit accidentally.

KEY FOR FLUSH

This key may be pressed while the system is sending output to the console. The first time it is pressed, output is no longer displayed and will be ignored (flushed) until FLUSH is pressed again. This can be done any number of times; FLUSH functions as a toggle. Note that processing continues while the output is ignored, so using FLUSH causes output to be lost.

KEY FOR STOP

This key may be pressed while the system is writing to CONSOLE. Like FLUSH, it is a toggle. Pressing it once causes output and processing to stop; pressing it again causes output and processing to resume; and so on. No output is lost; STOP is useful for slowing down a program so the output can be read while it is being sent to the terminal.

KEY TO ALPHA LOCK

When sent to the display unit, this character locks the keyboard in uppercase (alpha mode). It is usually a key on the keyboard as well.

KEY TO DELETE CHARACTER

This deletes the character where the cursor is and moves the cursor one character to the left.

KEY TO DELETE LINE

This key deletes the line that the cursor is currently on.

KEY TO END FILE

This key sets the intrinsic Boolean function EOF to TRUE when pressed while reading from the system input files (either KEYBOARD or INPUT, which come from device CONSOLE:).

KEYS TO MOVE CURSOR

The keys to move the cursor down, left, right, and up are recognized by the Screen-Oriented Editor, and are used when editing a document to move the cursor about the display unit. If your keyboard has a vector pad, you should use those keys for these functions. If you have no vector pad, you might select four keys in the same pattern (for example, ., K, ;, and O, in that order) and use them as your vector keys, prefixing them or using the corresponding ASCII control codes.

LEAD IN FROM KEYBOARD

Pressing certain keys generates a two-character sequence. The first character in these cases must always be a prefix and must be the same for all such sequences. This data item specifies that prefix. Note that this character is only accepted as a lead-in for characters where you have set PREFIXED[<item name>] to TRUE. (See MOVE CURSOR HOME for an example of this.)

LEAD IN TO SCREEN

Some terminals require a two-character sequence to activate certain functions. If the first character in all these sequences is the same, this data item can specify this prefix. This item is similar to the one above.

MAX NUMBER OF SUBSIDIARY VOLS

This field indicates the maximum number of subsidiary volumes that may be online at once. Because the p-System Unit Table expands a few bytes with each additional subsidiary volume entry, set this number to the smallest convenient value. (Also see FIRST SUBSIDIARY VOL NUMBER.)

The highest subsidiary volume will be FIRST SUBSIDIARY VOL NUMBER + MAX NUMBER OF SUBSIDIARY VOLS. This expression must be less than or equal to 127, which is the highest device number allowed for system units.

MAX NUMBER OF USER SERIAL VOLS

User-defined subsidiary volumes are not available on the Texas Instruments Professional Computer. This field should always be 0.

MOVE CURSOR HOME

When sent to the terminal, this key moves the cursor to the upper left of the display unit (position (0,0)). If your terminal does not have a character that does this, this data item must be set to CARRIAGE RETURN; then, you will not be able to use the Screen-Oriented Editor.

MOVE CURSOR RIGHT, LEFT, UP, and DOWN

When sent to the terminal, these move the cursor nondestructively one row or column. If your terminal does not have these functions, you will not be able to use the Screen-Oriented Editor.

NONPRINTING CHARACTER

This character is displayed on the display unit when a nonprinting character is typed or sent to the terminal while using the Screen-Oriented Editor.

PREFIXED [<item name>]

If you set this to TRUE, the system recognizes that a two-character sequence must be generated by a key or sent to the display unit for <item name>. See the explanations for LEAD IN FROM KEYBOARD and LEAD IN TO SCREEN. Note that one of these items is PREFIX[DELETE CHARACTER]. This refers to backspace; you can think of it as PREFIX[BACKSPACE].

PRINTABLE CHARACTERS

This entry is used to determine which character codes will be echoed to the console. Any code, from 0 to 255, may be echoed.

SETUP requires input in the form of a list of decimal values separated by commas or double periods. The values separated by commas correspond to the ASCII characters that will be echoed to the console. The double periods indicate that all values between the two indicated numbers are included; for example, 32 through 126 includes the values 32, 126, and all values between them. The typical values will be 13 and 32 through 126 (Carriage return is 13, and 32 through 126 are the standard printable characters). The value 13 must always be present.

SCREEN HEIGHT

Starting from 1, this is the number of lines in your display unit.

SCREEN WIDTH

Starting from 1, this is the number of characters in one line on your display.

SEGMENT ALIGNMENT

For ease of implementation, some systems require a code segment to be aligned to a certain address. For example, on 8086 based systems each code segment's starting address must be an integral multiple of 16 (for example, 0, 16, 32, and so on). Therefore, the segment alignment is 16. Most systems require no segment alignment and a value of 0 or 1 indicates this.

STUDENT

On all systems, this should be FALSE.

VERTICAL MOVE DELAY

This may be a decimal integer from 0 to 10. Many terminals require a delay after vertical cursor movements. This delay allows the movement to be completed before another character is sent. This data item specifies the number of nulls the system sends to the terminal after every CARRIAGE RETURN, ERASE TO END OF LINE, ERASE TO END OF SCREEN, CLEAR SCREEN, and MOVE CURSOR UP.

Summary of Data Items

All the fields which SETUP modifies are:

BACKSPACE
CODE POOL BASE[FIRST WORD]
CODE POOL BASE[SECOND WORD]
CODE POOL SIZE
EDITOR ACCEPT KEY
EDITOR ESCAPE KEY
EDITOR EXCHANGE-DELETE KEY
EDITOR EXCHANGE-INSERT KEY
ERASE LINE
ERASE SCREEN
ERASE TO END OF LINE
ERASE TO END OF SCREEN
FIRST SUBSIDIARY VOL NUMBER
HAS 8510A
HAS BYTE FLIPPED MACHINE
HAS CLOCK
HAS EXTENDED MEMORY
HAS LOWER CASE
HAS RANDOM CURSOR ADDRESSING
HAS SLOW TERMINAL
HAS SPOOLING
HAS WORD ORIENTED MACHINE
KEYBOARD INPUT MASK

KEY FOR BREAK
KEY FOR FLUSH
KEY FOR STOP
KEY TO ALPHA LOCK
KEY TO DELETE CHARACTER
KEY TO DELETE LINE
KEY TO END FILE
KEY TO MOVE CURSOR DOWN
KEY TO MOVE CURSOR LEFT
KEY TO MOVE CURSOR RIGHT
KEY TO MOVE CURSOR UP
LEAD IN FROM KEYBOARD
LEAD IN TO SCREEN
MAX NUMBER OF SUBSIDIARY VOLS
MAX NUMBER OF USER SERIAL VOLS
MOVE CURSOR HOME
MOVE CURSOR RIGHT
MOVE CURSOR UP
NONPRINTING CHARACTER
PREFIXED[DELETE CHARACTER]
PREFIXED[EDITOR ACCEPT KEY]
PREFIXED[EDITOR ESCAPE KEY]
PREFIXED[EDITOR EXCHANGE-DELETE KEY]
PREFIXED[EDITOR EXCHANGE-INSERT KEY]
PREFIXED[ERASE LINE]
PREFIXED[ERASE SCREEN]
PREFIXED[ERASE TO END OF LINE]
PREFIXED[ERASE TO END OF SCREEN]
PREFIXED[KEY TO DELETE CHARACTER]
PREFIXED[KEY TO DELETE LINE]
PREFIXED[KEY TO MOVE CURSOR DOWN]
PREFIXED[KEY TO MOVE CURSOR LEFT]
PREFIXED[KEY TO MOVE CURSOR RIGHT]
PREFIXED[KEY TO MOVE CURSOR UP]
PREFIXED[MOVE CURSOR HOME]
PREFIXED[MOVE CURSOR RIGHT]
PREFIXED[MOVE CURSOR UP]
PREFIXED[NON PRINTING CHARACTER]
PRINTABLE CHARACTERS
SCREEN HEIGHT

SCREEN WIDTH
SEGMENT ALIGNMENT
STUDENT
VERTICAL MOVE DELAY

Sample Terminal Setup

Here is a list of SYSTEM.MISCINFO data items followed by the default settings for the Texas Instruments Professional Computer.

Setup Field	Value	Comment
BACKSPACE	BS	May be modified by user
CODE POOL BASE[FIRST WORD]	00	See discussion of extended
CODE POOL BASE[SECOND WORD]	00	memory
CODE POOL SIZE	00	"
EDITOR ACCEPT KEY	ETX	May be modified by user
EDITOR ESCAPE KEY	ESC	May be modified by user
EDITOR EXCHANGE-DELETE KEY	112	DEL, may be modified
EDITOR EXCHANGE-INSERT KEY	111	INS, may be modified
ERASE LINE	76	Required
ERASE SCREEN	69	Required
ERASE TO END OF LINE	75	Required
ERASE TO END OF SCREEN	74	Required
FIRST SUBSIDIARY VOL NUMBER	12	Must be 12 or greater
HAS 8510A	FALSE	Required
HAS CLOCK	TRUE	May be modified by user
HAS BYTE FLIPPED MACHINE	FALSE	Required
HAS EXTENDED MEMORY	FALSE	May be modified by user
HAS LOWER CASE	TRUE	May be modified by user
HAS RANDOM CURSOR ADDRESSING	TRUE	Required
HAS SLOW TERMINAL	FALSE	Required
HAS SPOOLING	FALSE	May be modified by user
HAS WORD ORIENTED MACHINE	FALSE	Required
KEYBOARD INPUT MASK	255	Required
KEY FOR BREAK	255	Shift Break/Pause
		May be modified by user
KEY FOR FLUSH	06	CTRL F, required
		May be modified by user
KEY FOR STOP	19	CTRL S, may be modified
KEY TO ALPHA LOCK	NUL	May be modified by user
KEY TO DELETE CHARACTER	BS	May be modified by user
KEY TO DELETE LINE	DEL	Control Backspace
KEY TO END FILE	ETX	May be modified by user
KEY TO MOVE CURSOR DOWN	66	May be modified
KEY TO MOVE CURSOR LEFT	68	May be modified
KEY TO MOVE CURSOR RIGHT	67	May be modified
KEY TO MOVE CURSOR UP	65	May be modified

Setup Field	Value	Comment
LEAD IN FROM KEYBOARD	17	Required
LEAD IN TO SCREEN	ESC	Required
MAX NUMBER OF SUBSIDIARY VOLS	00	May be modified by user
MAX NUMBER OF USER SERIAL VOLS	10	Required
MOVE CURSOR HOME	72	Required
MOVE CURSOR RIGHT	67	Required
MOVE CURSOR UP	65	Required
NONPRINTING CHARACTER	63	May be modified by user
PREF[DELETE CHARACTER]	FALSE	May be modified by user
PREF[EDITOR ACCEPT KEY]	FALSE	May be modified by user
PREF[EDITOR ESCAPE KEY]	FALSE	May be modified by user
PREF[EDITOR EXCHANGE-DELETE KEY]	TRUE	May be modified by user
PREF[EDITOR EXCHANGE-INSERT KEY]	TRUE	May be modified by user
PREF[ERASE LINE]	TRUE	Required
PREF[ERASE SCREEN]	TRUE	Required
PREF[ERASE TO END OF LINE]	TRUE	Required
PREF[ERASE TO END OF SCREEN]	TRUE	Required
PREF[KEY TO DELETE CHARACTER]	FALSE	May be modified by user
PREF[KEY TO DELETE LINE]	FALSE	May be modified by user
PREF[KEY TO MOVE CURSOR DOWN]	TRUE	May be modified by user
PREF[KEY TO MOVE CURSOR LEFT]	TRUE	May be modified by user
PREF[KEY TO MOVE CURSOR RIGHT]	TRUE	May be modified by user
PREF[KEY TO MOVE CURSOR UP]	TRUE	May be modified by user
PREF[MOVE CURSOR HOME]	TRUE	Required
PREF[MOVE CURSOR RIGHT]	TRUE	Required
PREF[MOVE CURSOR UP]	TRUE	Required
PREF[NONPRINTING CHAR]	FALSE	Required
PRINTABLE CHARACTERS	13, 32 ...126	Required
SEGMENT ALIGNMENT	16	Required
SCREEN HEIGHT	25	Required
SCREEN WIDTH	80	May be modified by user
STUDENT	FALSE	Required
VERTICAL MOVE DELAY	00	Required

PRINT SPOOLING

The print spooler is a program that allows you to queue and print files concurrently with the normal execution of the p-System (while the console is waiting for input from the keyboard). The queue it creates is a file called *SYSTEM.SPOOLER, and the files you wish to print must reside on volumes that are online or an error will occur.

When SPOOLER is X(ecuted, the following menu appears:

```
Spool: P(rint, D(elete, L(ist, S(uspend, R(esume, A(bort,
  C(lear, Q(uit
```

The following paragraphs explain the items on this menu:

P(rint — Prompts for the name of a file to be printed. This name is then added to the queue. If SYSTEM.SPOOLER does not already exist, it is created. In the simplest case P(rint can be used to send a single file to the printer. Up to 21 files can be placed in the print queue.

D(elete — Prompts for a file name to be taken out of the print queue. All occurrences of that file name are taken out of the queue.

L(ist — Displays the files currently in the queue.

S(uspend — Temporarily halts printing of the current file.

R(esume — Continues printing the current file after a S(uspend. R(esume also starts printing the next file in the queue after an error or an A(bort.

A(bort	Permanently stops the printing process of the current file and takes it out of the queue.
C(lear	Deletes all file names from the queue.
Q(uit	Exits the spooler utility and starts transferring files to the printer.

If an error occurs (for example, a nonexistent file is specified in the queue), the error message appears only when the p-System is at the command menu. If necessary, the spooler waits until you return to the outer level.

Program output to the printer can run concurrently with spooler output. The spooler finishes the current file and then turns the printer over to your program. (Your program is suspended while it waits for the printer.) Your program should only do Pascal (or other high-level) writes to the printer. If your program does printer output using unitwrite, the output is sent immediately and appears randomly interspersed with the spooler output.

The utility SPOOLER.CODE uses the operating system unit SPOOLOPS. Within this unit is a process called spooltask. Spooltask is started at boot time and runs concurrently with the rest of the UCSD p-System. The print spooler automatically restarts at boot time if *SYSTEM.SPOOLER is not empty. When the file *SYSTEM.SPOOLER exists, spooltask prints the files that it names. Spooltask runs as a background to the main operations of the p-System.

*SPOOLER.CODE interfaces with SPOOLOPS and uses routines within it to generate and alter the print queue within *SYSTEM.SPOOLER.

To restart the print spooling process if SPOOLER.CODE is executing when the system goes down, reboot the system, press X(ecute from the command prompt line, enter *SPOOLER.CODE, and press the **RETURN** key. Then press R(esume.

A

Bootstrapping the UCSD p-System

In order to start the p-System running (or bootstrap it) on the Texas Instruments Professional Computer, you should turn on the system's power and insert the PSYS: disk into drive #4: (the left drive). You can manually restart the p-System at any time by striking the ALT, DEL, and CTRL keys simultaneously.

The rest of the information in this appendix concerns the details of the bootstrap process and the various errors that might occur during it.

The p-System boot process includes four steps: the ROM boot, the interpreter boot, the system boot, and system initialization.

When you bootstrap or restart (reboot) the p-System, the computer performs initialization and self-tests. The computer then loads the boot sectors, track 0 sectors 0 and 1, into memory. This code then further initializes the system as required to execute the p-System and loads the hardware dependent routines, SYSTEM.INTERP, into memory.

If extended memory is present (more than 64K) and has not already been initialized, a RAM disk directory is built. The number of blocks specified when the RAM Configuration Utility was run are copied from the boot disk onto the RAM disk. The RAM disk is searched for the file SYSTEM.PASCAL and SYSTEM.MISCINFO. If found, the system boots from the RAM disk, unit #11. Otherwise, the boot procedure continues from the disk in drive #4.

SYSTEM.INTERP begins execution by loading the operating system, SYSTEM.PASCAL, into memory. (SYSTEM.PASCAL consists of p-code which is interpreted by SYSTEM.INTERP to provide the system functions.)

At this point, the system boot process is complete and control is handed to the operating system. However, the system initialization is not yet complete. SYSTEM.PASCAL causes the file SYSTEM.MISCINFO which contains information specific to the Texas Instruments Professional Computer, to be read. Based on the contents of this file, the operating system defines environmental parameters required for efficient system operation on the Texas Instruments Professional Computer.

As indicated by the boot procedure, the following files are required on each bootable system disk:

SYSTEM.INTERP
SYSTEM.PASCAL
SYSTEM.MISCINFO

In addition, the system disk must contain a correct Texas Instruments p-System boot loader installed on track 0 sectors 0 and 1. (Refer to the description of the Disk Formatting Utility in Chapter 5.) If the system is tailored to boot from the RAM disk, the file SYSTEM.INTERP and the boot sectors are still loaded from the disk in drive #4: and are not required on the RAM disk volume. All other system files required for booting must be present on the RAM disk for the boot procedure to complete correctly.

During this boot procedure, several errors may occur. The following table summarizes the possible errors and user steps to correct the error.

Error Message	Probable Cause/Solution
** System Error ** - 00xx on Drive n	Disk error encountered while reading the system disk. Refer to the following table for a description of the error code xx.
Unable to locate SYSTEM.INTERP	The required file was not located on the boot drive. Copy the indicated file onto the disk and try again.
Unable to locate SYSTEM.PASCAL	The required file was not located on the boot drive. Copy the indicated file onto the disk and try again.
Error reading SYSTEM.INTERP	A disk error caused the boot process to terminate prematurely. This indicates either a bad disk or drive. Try again with a backup disk. If the error persists, service the drive.
Error reading SYSTEM.PASCAL	A disk error caused the boot process to terminate prematurely. This indicates either a bad disk or drive. Try again with a backup disk. If the error persists, service the drive.

If an error is encountered, the error message is displayed and the following prompt is provided to allow you to acknowledge the error.

```
* Please insert system disk, and
* Strike any key when ready...
```

Once a key has been struck, the boot process restarts. If a disk error was encountered, the boot begins by searching the remaining drives, if any, for a system disk. Otherwise, the boot tries again with the disk which encountered the error.

If the message ** System Error ** - 00xx on Drive n is displayed, some error was encountered while reading the system disk. In this message, n is either A or B indicating the drive from which the error was received, and xx is one of the following error codes:

Code xx	Probable Description	Cause/Solution
30	No drives installed	System disk system failure. Ensure that system hardware is properly installed. If error persists, system may require servicing.
31	Disks not ready	No system disk in any drive. Place bootable system disk in drive.
32	Disk error	Error reading disk.
33		Retry with backup.
34		If error persists, system may require servicing.
36	Not a TI system disk	The system disk does not contain a valid TI system disk. Retry with a valid TI system disk.
37	Disk format error	The disk has not been properly formatted. Retry with a valid TI system disk.
38	Bad boot sector CRC	The boot sector read operation received data errors. Retry with a backup. If failure persists, system may require servicing.
39	Controller failure	The disk system controller has failed. The system requires servicing.

B

Special Keys

p-System	Texas Instruments Professional Computer
BACKSPACE	BACKSPACE
TAB	TAB
UP ARROW	Marked accordingly
DOWN ARROW	Marked accordingly
LEFT ARROW	Marked accordingly
RIGHT ARROW	Marked accordingly
RETURN	RETURN
ETX	CTRL-C
ESC	ESC
BREAK	Shifted BREAK/PAUSE
STOP	Unshifted BREAK/PAUSE or CTRL-S
FLUSH	CTRL-F
DELETE LINE	CTRL-BACKSPACE
EXCHANGE-INSERT	INS
EXCHANGE-DELETE	DEL
ALPHA-LOCK	CAPS LOCK

Execution Errors

0	Fatal system error
1	Invalid index, value out of range
2	No segment, bad code file
3	Procedure not present at exit time
4	Stack overflow
5	Integer overflow
6	Divide by zero
7	Invalid memory reference <bus timed out>
8	User break
9	Fatal system I/O error
10	User I/O error
11	Unimplemented instruction
12	Floating point math error
13	String too long
14	Halt, breakpoint
15	Bad block
16	Breakpoint
17	Incomplete real number size
18	Set too large
19	Segment too large

All run-time errors cause the system to I(nitialize itself; fatal errors cause the system to re-bootstrap. Some fatal errors leave the system in an irreparable state, in which case the user must re-bootstrap.

D

I/O Results

0	No error
1	Bad block, parity error (CRC)
2	Bad device number
3	Illegal I/O request
4	Data-com timeout
5	Volume is no longer on-line
6	File is no longer in directory
7	Bad file name
8	No room, insufficient space on volume
9	No such volume on-line
10	No such file on volume
11	Duplicate directory entry
12	Not closed: attempt to open an open file
13	Not open: attempt to access a closed file
14	Bad format: error in reading real or integer
15	Ring buffer overflow
16	Volume is write-protected
17	Illegal block number
18	Illegal buffer

Device Number Assignments

The p-System on the Texas Instruments Professional Computer supports up to 4 disk drives, a RAM disk, a parallel printer, the serial communications option card, and up to 10 subsidiary volumes. The following table describes the shipped unit number assignments as defined by SYSTEM.MISCINFO.

Device Number	Volume Name	Comment
1	CONSOLE	
2	SYSTERM	
4	DS01	Volume name assigned by user
5	DS02	Volume name assigned by user
* 6	PRINTER	Optional printer required
* 7	REMIN	Optional communications required
* 8	REMOUT	Optional communications required
* 9	DS03	Optional disk drive required
*10	DS04	Optional disk drive required
*11	RAM	Optional memory required volume name may be assigned by user
*12		First user subsidiary vol.
...		
*21		Last user subsidiary vol.

Support for optional devices is included in standard device support. However, the device is not required for system operation.

F

ASCII Codes

Decimal	Octal	Hexadecimal	Character
0	000	00	NUL
1	001	01	SOH
2	002	02	STX
3	003	03	ETX
4	004	04	EOT
5	005	05	ENQ
6	006	06	ACK
7	007	07	BEL
8	010	08	BS
9	011	09	HT
10	012	0A	LF
11	013	0B	VT
12	014	0C	FF
13	015	0D	CR
14	016	0E	SO
15	017	0F	SI
16	020	10	DLE
17	021	11	DC1
18	022	12	DC2
19	023	13	DC3
20	024	14	DC4
21	025	15	NAK
22	026	16	SYN
23	027	17	ETB
24	030	18	CAN
25	031	19	EM
26	032	1A	SUB
27	033	1B	ESC
28	034	1C	FS
29	035	1D	GS
30	036	1E	RS
31	037	1F	US
33	041	21	!
34	042	22	"
35	043	23	#

Decimal	Octal	Hexadecimal	Character
36	044	24	$
37	045	25	%
38	046	26	&
39	047	27	'
40	050	28	(
41	051	29	)
42	052	2A	*
43	053	2B	+
44	054	2C	,
45	055	2D	-
46	056	2E	.
47	057	2F	/
48	060	30	0
49	061	31	1
50	062	32	2
51	063	33	3
52	064	34	4
53	065	35	5
54	066	36	6
55	067	37	7
56	070	38	8
57	071	39	9
58	072	3A	:
59	073	3B	;
60	074	3C	<
61	075	3D	=
62	076	3E	>
63	077	3F	?
64	100	40	@
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4A	J
75	113	4B	K
76	114	4C	L

Decimal	Octal	Hexadecimal	Character
77	115	4D	M
78	116	4E	N
79	117	4F	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5A	Z
91	133	5B	[
92	134	5C	/88
93	135	5D	]
94	136	5E	^
95	137	5F	_
96	140	60	\
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u

Decimal	Octal	Hexadecimal	Character
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z
123	173	7B	{
124	174	7C	|
125	175	7D	}
126	176	7E	~
127	177	7F	DEL

G

Extended Memory on the Texas Instruments Professional Computer

There are two features supported by the p-System on the Texas Instruments Professional Computer that allow you to utilize up to 256K bytes of installable memory. The Extended Memory option allows you to allocate up to 64K of additional memory to the execution of programs. This allows for larger program segments and potentially less disk access overhead. The RAM Disk option allows you to designate up to 192K bytes of memory to an in-memory or RAM disk. This allows placing often-accessed files in memory, eliminating much disk access and providing greater throughput.

The Extended Memory option must be enabled by the RAM Configuration Utility provided on the PSYS disk. You may specify that as much as 64K or as little as 32K be allocated to execution of user and system programs.

The RAM disk is always enabled, provided that sufficient system memory exists. It is present in any system configuration if your Texas Instruments Professional Computer has more than 64K bytes of RAM installed and all of the additional memory is not allocated to the extended memory codepool. The number of blocks available in the RAM disk may be determined using the V(olume option of the F(iler command. The following display depicts the V(olume listing of a typical system configuration with 192K bytes of memory. The RAM disk is always unit #11, and the default volume name is always RAM.

```
Vols on-line:
  1    CONSOLE:
  2    SYSTERM:
  4 #  PSYS:          [ 640]
  6    PRINTER:
  7    REMIN:
  8    REMOUT:
 11 #  RAM:           [ 128]
Root vol is - PSYS:
Prefix is   - PSYS:
```

The RAM disk is treated like any other storage volume, and the system utilities and user programs may access it as they would a flexible diskette, by either device number or volume name.

The Z(ero command of the filer may be used to reinitialize the RAM disk and to assign it a new volume name. When using the Z(ero command, you should never exceed the default provided for the number of blocks, since this value is the maximum number which may be contained in the extended memory available in the system configuration.

When the system boots, it inspects the memory location set aside for the RAM disk to determine if a valid directory is present. If there is a directory, the RAM disk is not altered. (This means that you can reboot and still have the same RAM disk available, as long as you do not turn the power off.) If there is not a current RAM disk directory, one is created and the RAM disk is zeroed. In this case, the system copies each file from the system disk to the RAM disk until the file named RAM.INIT is encountered. (This is a one block file that you should create with the filer's M(ake activity.) No files are moved from the system disk unless all of the files located before the RAM.INIT fit in the available RAM disk space. If RAM.INIT is not found on the system disk, no files are moved to the RAM disk.

For example, if your system disk has these files:

```
PSYS:
SYSTEM.PASCAL     106  1-Jan-83
SYSTEM.MISCINFO     1  1-Jan-83
SYSTEM.FILER       38  1-Jan-83
SYSTEM.STARTUP      8  1-Jan-83
RAM.INIT            1  1-Jan-83
SYSTEM.EDITOR      47  1-Jan-83
SYSTEM.LIBRARY     18  1-Jan-83
```

and your RAM disk contains 256 blocks, the files SYSTEM.PASCAL, SYSTEM.MISCINFO, SYSTEM.FILER, and SYSTEM.STARTUP are copied to the newly initialized RAM disk.

If, after this process, the RAM disk contains the files SYSTEM.PASCAL and SYSTEM.MISCINFO, the system continues the boot procedure from the RAM disk. Once the RAM disk is initialized, it remains useable until power is removed from the system. If it becomes necessary to reboot, reboot by pressing the **CTRL**, **ALT** and **DEL** keys simultaneously (rather than turning the power off and on). This preserves any data located on the RAM disk.

The p-System, utilizes any memory above 64K bytes as RAM disk. The first 64K bytes are used for executable code and the internal workings of the p-System. See "RAM Configuration Utility" (Chapter 5) for more information about of this.

Release Disk Configurations

File Name	PSYS	PDEV
SYSTEM.PASCAL	X	
SYSTEM.INTERP	X	
SYSTEM.MISCINFO	X	
SYSTEM.FILER	X	
SYSTEM.EDITOR	X	
SPOOLER.CODE	X	
SYSTEM.LIBRARY	X	
SYSTEM.SYNTAX	X	
SYSTEM.FONT	X	
COMPRESS.CODE	X	
SETUP.CODE	X	
PATCH.CODE	X	
LIBRARY.CODE	X	
COPYDUPDIR.CODE	X	
MARKDUPDIR.CODE	X	
RECOVER.CODE	X	
FORMAT.CODE	X	
DECODE.CODE	X	
SETTIME.CODE	X	
CONFIG.RAM.CODE	X	
CONFIG.REM.CODE	X	
CONFIG.PTR.CODE	X	
CONFIG.PTR.TEXT	X	
REALCONV.CODE	X	
ABSWRITE.CODE	X	
BOOT.CODE	X	
DISKSIZE.CODE	X	
PRINT.CODE	X	
SYSTEM.LINKER		X
SYSTEM.ASSEMBLER		X
SYSTEM.COMPILER		X
8086.OPCODES		X
8086.ERRORS		X
8087.FOPS		X

File Name	PSYS	PDEV
XREF.CODE		X
BINDER.CODE		X
COMMANDIO.CODE		X
SCREENOPS.CODE		X
KERNEL.CODE		X
SPOOLOPS.CODE		X
WILD.CODE		X
DIR.INFO.CODE		X
FILE.INFO.CODE		X
SYS.INFO.CODE		X
ERRORHANDL.CODE		X

Glossary

adaptable system — a version of the p-System that is structured so that you only need to write a new SBIOS (rather than an entire BIOS) to bring the system up. The p-machine emulator must already be written for the processor that the computer uses.

assembler — the p-System component which translates human-readable assembly language into machine code. The p-System has assemblers for these processors: 8080, Z80, 6502, LSI-11, PDP-11, 9900, 6809, 8086/8088, 68000, 6800 and Z8.

back file — a backup file for text files that is identified by the suffix .BACK, which is appended to the file name; for example, FILENAME.BACK.

bad file — an immobile file used to prevent the use of damaged portions of a disk. A bad file is identified by the suffix .BAD, which is appended to the file name; for example, BAD.00120.BAD.

BASIC — a popular high level programming language which is accepted by one of the p-System's compilers.

BIOS — basic input output subsystem; that portion of a p-machine emulator that is specific to a particular configuration of devices on a computer.

block-structured device — see storage volume.

bootstrap — the action of starting (or that piece of code which starts) the p-System running. You must bootstrap the p-System before you can do anything with it.

boot volume — see system disk.

chaining — see program chaining.

client — a program or unit which uses another unit.

code file — a file that contains compiled or assembled code that is identified by the suffix .CODE; for example, FILENAME.CODE.

code segment — a portion of code which may be swapped in and out of memory during execution. A program or unit may be divided into several segments.

command menu — the main p-System menu. It is displayed by the operating system.

communications volume — any I/O device which does not store information permanently (such as the console or printer).

compiled listing — a collection of detailed information produced by the compiler about a program.

compilation unit — the resultant code file or code segments from a single compilation. It may include one or more units and/or a program segment.

compiler — a translator which accepts a high level language program as input and outputs an executable p-code object codefile.

control keys — see special keys.

data file — a file that contains user data. It can be in any format. It has no special file name suffix.

decode — a utility which allows you to see the inner details of a code file.

default disk — the volume where the p-System assumes files reside unless you specifically indicate otherwise.

device — any peripheral equipment that is a recipient or producer of data. A device needs a driver program to communicate with the operating system. Devices are classified into storage devices (disks) and communication devices (console, printer, remote).

device number — a number used to refer to a particular storage or communications volume. It is always preceded by a number sign (#) and usually followed by a colon (:). For example, #5:.

directory — an area on a storage volume which contains information about the files which reside there (such as their names and locations).

directory listing — a display, usually on the console, of the files on a given storage volume.

editor — one of the three p-System text manipulation utilities. These are the Screen-Oriented Editor, EDVANCE, and YALOE. With an editor, you can create, examine, and alter text files.

execution error — an error which can occur when a running program does something incorrect. The program then terminates and the p-System is reinitialized.

extended memory — a feature on several processors that allows the user to access more than 64K of main memory.

file — a collection of information that resides on a disk and can be brought into main memory when needed by the p-System or a program. It may contain code, text, a subsidiary volume, a graphic image, or data.

FORTRAN-77 — a popular high-level programming language accepted by one of the p-System's compilers.

foto file — a file that contains one graphic image for use by the Turtlegraphics routines. It ends with the suffix .FOTO. For example: PICTURE.FOTO.

interleaving — the process of mapping the physical sectors to the logical sector number assignments within one track of a disk. It is part of the disk formatting process. Interleaving optimizes disk performance for consecutive read or write accesses.

I/O — input and output.

I/O error — a problem that can occur, under several circumstances, when something goes wrong with an I/O operation. For example, a disk write will fail if the disk has been inappropriately removed from its drive.

I/O redirection — a feature that allows the p-System's input (or a program's input) to come from some place other than the keyboard. Also, output for the p-System and programs can be sent to some place other than the screen.

interpreter — see p-machine emulator.

libraries — code files that contain one or more units which can be used by programs and other units.

library utility — a program that allows you to place code segments from different code files into one code file. Units can be placed in a library using this utility.

linker — a p-System component that combines assembled code files together. It can also combine a compiled code file with assembled code files.

long integer — a special facility that allows programs to use integer arithmetic with up to 36 decimal digits of precision.

marker — an invisible flag that marks a particular location within a text file.

menu — a list of available commands or options that are displayed on the screen by the operating system and major p-System components. A command or option can be selected from a menu with a single keystroke.

monitor — a command menu option which can cause the system to keep track of all the keystrokes that you enter at the keyboard. The result is stored in a disk file which can later be the source of I/O redirection.

mount — to cause a subsidiary volume to be accessible to the p-System.

multitasking — the execution of two or more tasks concurrently. A task is a piece of code called a process in UCSD Pascal.

native code — machine level code that is produced by the native code generator or is written by a programmer in assembly language.

n-code — see native code.

native code generator — a program that translates portions of an executable p-code file into n-code. The resulting code file always contains a combination of p-code and n-code.

nonblock-structured device — see communications volumes.

on-line — the status of a volume when the p-System can access it. For a storage volume to be on-line, the disk must be in the appropriate drive. For a communications volume to be on-line, the I/O device must be properly connected and turned on.

Pascal — see UCSD Pascal.

p-code — psuedo code: p-machine code generated by the p-System compilers and emulated by the p-machine emulators.

p-machine — an idealized pseudo computer optimized for high level language execution on small host machines which execute p-code.

p-machine emulator — the part of the p-System that allows microcomputer hardware to imitate the operation of the p-machine. It is written in assembly language.

PME — see p-machine emulator.

p-System — see UCSD p-System.

portability — the capability of moving executable code written on one microcomputer to another without recompilation. This is possible because programs are compiled into p-code which can be executed on any computer which hosts the UCSD p-System.

prefix, file — a volume name appended to the beginning of a file name. It indicates the volume on which the file resides. A colon separates the prefix from the file name; for example, VOLNAME:FILENAME.TEXT. If no prefix is specified, the default disk is used.

program chaining — the process of calling for the execution of one program from another program without relinquishing control back to the user. The p-System allows an arbitrary number of programs to be chained together.

print spooler — a program that allows you to queue and print as many as 20 text files concurrently with normal execution of the p-System.

prompt — a question, displayed on the screen, asking you for information. You respond to a prompt by entering the information and pressing the **RETURN** key.

reboot — to start up the p-System again. To rebootstrap.

root volume — see system disk.

SBIOS — simplified basic input output subsystem. A portion of the BIOS on adaptable systems. The SBIOS of an adaptable system must be rewritten for each different computer on which the adaptable system is run.

Screen-Oriented Editor — a text editing program designed for operation with an interactive console.

segment — see code segment.

special keys — the keys that have a particular meaning to the p-System (other than indicating an ordinary character). For example, the **RETURN** key indicates a carriage return in the editor.

storage volume — a disk or subsidiary volume. Any I/O device which stores information permanently.

subsidiary volumes — a file organization facility that provides two levels of directory hierarchy. The subsidiary volume is located within a .SVOL file on the principal volume.

suffix, file — one of several special sequences of characters appended to the end of a file name after a period. The file suffix usually indicates the file type. The standard file suffixes are .TEXT, .CODE, .SVOL, .BACK, .BAD, and .FOTO.

.SVOL file — a file identified by the suffix .SVOL that contains a subsidiary volume; for example, NAME.SVOL.

system disk — the disk which was bootstrapped. It contains the operating system software. It can be represented to the p-System by the asterisk (*).

system files — the disk files which contain the main components of the UCSD p-System.

syntax error — an error that occurs when a compiler detects something incorrect in the program text.

text file — a file that contains user-readable character information (as opposed to machine code), identified by the suffix .TEXT; for example, FILENAME.TEXT.

Turtlegraphics — a package of routines that create and manipulate images on a graphic display.

UCSD — University of California at San Diego.

UCSD Pascal — a programming language which is a slightly extended version of standard Pascal.

UCSD p-System — portable microcomputer software environment for execution and development of applications programs (sometimes called the Universal Operating System™*).

unit — a collection of routines and associated data structures which is compiled as a whole. Routines within a unit may be used by programs or other units.

Universal Medium™* — a 5-1/4 inch diskette format that is used by several computers. It facilitates the physical transport of data between them.

utilities — programs that assist you in various areas of p-System use by doing such tasks as developing programs, maintaining files, printing files, and so forth.

volume — either a storage volume (disk or subsidiary volume) or a communications volume (such as the console or printer).

volume ID — the name or device number for a volume (either storage or communications volume).

* Universal Operating System and Universal Medium are trademarks of SofTech Microsystems, Incorporated.

volume name — seven or fewer characters, followed by a colon (:) which refer to a particular storage or communications volume.

work file — scratch-pad text and/or code files that you can use when developing programs. A work file may be created by designating existing files, or by creating a new file with the Editor.

wild card characters — three special symbols ($, =, and ?) which allow the filer to operate on several files at one time.

Index

THREE-MONTH LIMITED WARRANTY TEXAS INSTRUMENTS PROFESSIONAL COMPUTER SOFTWARE MEDIA

TEXAS INSTRUMENTS INCORPORATED EXTENDS THIS CONSUMER WARRANTY ONLY TO THE ORIGINAL CONSUMER PURCHASER.

WARRANTY DURATION

The media is warranted for a period of three (3) months from the date of original purchase by the consumer.

Some states do not allow the exclusion or limitation of incidental or consequential damages or limitations on how long an implied warranty lasts, so the above limitations or exclusions may not apply to you.

WARRANTY COVERAGE

This limited warranty covers the cassette or diskette ("media") on which the computer program is furnished. It does not extend to the program contained on the media or the accompanying book materials (collectively the "Program"). The media is warranted against defects in material or workmanship. THIS WARRANTY IS VOID IF THE MEDIA HAS BEEN DAMAGED BY ACCIDENT, UNREASONABLE USE, NEGLECT, IMPROPER SERVICE OR OTHER CAUSES NOT ARISING OUT OF DEFECTS IN MATERIALS OR WORKMANSHIP.

PERFORMANCE BY TI UNDER WARRANTY

During the above three-month warranty period, defective media will be replaced when it is returned postage prepaid to a Texas Instruments Service Facility listed below or an authorized Texas Instruments Professional Computer Dealer with a copy of the purchase receipt. The replacement media will be warranted for three months from date of replacement. Other than the postage requirement (where allowed by state law), no charge will be made for the replacement. TI strongly recommends that you insure the media for value prior to mailing.

WARRANTY AND CONSEQUENTIAL DAMAGES DISCLAIMERS

ANY IMPLIED WARRANTIES ARISING OUT OF THIS SALE INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO THE ABOVE THREE MONTH PERIOD. TEXAS INSTRUMENTS SHALL NOT BE LIABLE FOR SPECIAL, COLLATERAL, INCIDENTAL OR CONSEQUENTIAL COSTS, EXPENSES, OR DAMAGES INCURRED BY THE CONSUMER OR ANY OTHER USER ARISING OUT OF THE PURCHASE OR USE OF THE MEDIA. THESE EXCLUDED DAMAGES INCLUDE, BUT ARE NOT LIMITED BY, COST OF REMOVAL OR REINSTALLATION, OUTSIDE COMPUTER TIME, LABOR COSTS, LOSS OF GOODWILL, LOSS OF PROFITS, LOSS OF SAVINGS, OR LOSS OF USE OR INTERRUPTION OF BUSINESS.

LEGAL REMEDIES

This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

TEXAS INSTRUMENTS CONSUMER SERVICE FACILITIES

U.S. Residents:

Texas Instruments Service Facility
P.O. Box 1444, MS 7758
Houston, Texas 77001

Canadian Residents:

Geophysical Service Inc.
41 Shelley Road
Richmond Hill, Ontario
Canada L4C 5G4

Consumers in California and Oregon may contact the following Texas Instruments offices for additional assistance or information.

Texas Instruments
Consumer Service
831 South Douglas St.
Suite 119
El Segundo, California 90245
(213) 973-2591

Texas Instruments
Consumer Service
6700 S.W. 105th
Kristin Square, Suite 110
Beaverton, Oregon 97005
(503) 643-6758

IMPORTANT NOTICE OF DISCLAIMER REGARDING THE PROGRAM

The following should be read and understood before using the software media and Program.

TI does not warrant that the Program will be free from error or will meet the specific requirements of the purchaser/user. The purchaser/user assumes complete responsibility for any decision made or actions taken based on information obtained using the Program. Any statements made concerning the utility of the Program are not to be construed as expressed or implied warranties.

TEXAS INSTRUMENTS MAKES NO WARRANTY, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE PROGRAM AND MAKES ALL PROGRAMS AVAILABLE SOLELY ON AN "AS IS" BASIS.

IN NO EVENT SHALL TEXAS INSTRUMENTS BE LIABLE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH OR ARISING OUT OF THE PURCHASE OR USE OF THE PROGRAM. THESE EXCLUDED DAMAGES INCLUDE, BUT ARE NOT LIMITED BY, COST OF REMOVAL OR REINSTALLATION, OUTSIDE COMPUTER TIME, LABOR COSTS, LOSS OF GOODWILL, LOSS OF PROFITS, LOSS OF SAVINGS, OR LOSS OF USE OR INTERRUPTION OF BUSINESS. THE SOLE AND EXCLUSIVE LIABILITY OF TEXAS INSTRUMENTS, REGARDLESS OF THE FORM OF ACTION, SHALL NOT EXCEED THE PURCHASE PRICE OF THE PROGRAM. TEXAS INSTRUMENTS SHALL NOT BE LIABLE FOR ANY CLAIM OF ANY KIND WHATSOEVER BY ANY OTHER PARTY AGAINST THE PURCHASER/USER OF THE PROGRAM.

COPYRIGHT

TEXAS INSTRUMENTS PROFESSIONAL COMPUTER
UCSD p-System™ Operating System Reference Manual
TI Part No. 2232395-0001

Original Issue: 15 April 1983

Your Name: _____

Company: _____

Telephone: _____

Department: _____

Address: _____

City/State/Zip Code: _____

Your comments and suggestions assist us in improving our products. If your comments concern problems with this manual, please list the page number.

Comments:

This form is not intended for use as an order blank.

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 6189 HOUSTON, TX

POSTAGE WILL BE PAID BY ADDRESSEE

Texas Instruments Incorporated
Attn: Marketing M/S 7896
P.O. Box 1444
Houston, TX 77001

FOLD

TEXAS INSTRUMENTS PROFESSIONAL COMPUTER
UCSD p-System™ Operating System Reference Manual
TI Part No. 2232395-0001

Original Issue: 15 April 1983

Your Name: _____

Company: _____

Telephone: _____

Department: _____

Address: _____

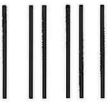
City/State/Zip Code: _____

Your comments and suggestions assist us in improving our products. If your comments concern problems with this manual, please list the page number.

Comments:

This form is not intended for use as an order blank.

FOLD



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 6189 HOUSTON, TX

POSTAGE WILL BE PAID BY ADDRESSEE

Texas Instruments Incorporated
Attn: Marketing M/S 7896
P.O. Box 1444
Houston, TX 77001

FOLD

TEXAS INSTRUMENTS PROFESSIONAL COMPUTER
UCSD p-System™ Operating System Reference Manual
TI Part No. 2232395-0001

Original Issue: 15 April 1983

Your Name: _____

Company: _____

Telephone: _____

Department: _____

Address: _____

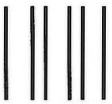
City/State/Zip Code: _____

Your comments and suggestions assist us in improving our products. If your comments concern problems with this manual, please list the page number.

Comments:

This form is not intended for use as an order blank.

FOLD



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 6189 HOUSTON, TX

POSTAGE WILL BE PAID BY ADDRESSEE

Texas Instruments Incorporated
Attn: Marketing M/S 7896
P.O. Box 1444
Houston, TX 77001

FOLD

Texas Instruments reserves the right to change its product and service offerings at any time without notice.

TEXAS INSTRUMENTS

2232395-0001

Printed in U.S.A.