SOFTECH
MICROSYSTEMS

p-System™ Software Reference Library

Operating System

1-100.41.A

p-System™

Operating System Reference Manual

SofTech Microsystems, Inc.
San Diego, California

1-100.41.A

Printed in the United States of America.

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.

PREFACE

This publication is a reference manual for the p-System® operating system, file manager, screen-oriented editor, and several utilities. It describes the facilities of these major p-System components and provides basic instructions for using them. If you are somewhat 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 the UCSD p-System

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

Program Development Reference Manual
Assembler Reference Manual
Optional Products Reference Manual
Internal Architecture Reference Manual
Adaptable System Installation Manual
UCSD Pascal Handbook
FORTRAN-77 Reference Manual
BASIC Reference Manual

All of these publications are available from SofTech Microsystems. Personal Computing with the UCSD p-System and UCSD Pascal Handbook are published by Prentice-Hall and are available in bookstores, as well.

T A B L E

O F

C O N T E N T S

Table of Contents

Table of Contents

Table of Contents

CHAPTER 1

INTRODUCTION

ORGANIZATION OF THIS MANUAL

This book is the main user reference manual for the p-System™.

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," describes the function of each system command. It also presents examples of the main system menu and gives suggestions and procedures for interacting with the p-System.

Chapter 3, "File Management," presents information about file organization and file handling, as well as descriptions of the file manager ("filer") and its functions.

Chapter 4, "Screen-Oriented Editor," describes the p-System's main text editor.

Chapter 5, "Utility Programs," covers several p-System utilities. These utilities can help you to print files, recover lost files, configure the p-System for a particular terminal, and so forth.

The appendices present useful reference material:

A Execution Errors
B I/O Results/Errors
C Device Numbers
D ASCII Code
E p-System Configuration Notes
F USUS Membership Application
G Software Problem Reporting
H p-System Glossary

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. This universal operating system now offers fully compatible, integrated compilers for UCSD Pascal®, FORTRAN-77, and BASIC. 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, 8080/8085, 8086/8088/8087, 6502, 6809, 68000, 9900, PDP-11, LSI-11, and VAX.

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 couldn't 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.

DESIGN PHILOSOPHY

The development team, 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:

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

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

3. 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 bascially 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.

Portability

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 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 screen 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 <backspace> if you make a mistake while responding to a prompt.

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

Figure 1-2 graphically describes the interrelationships of the major p-System components.

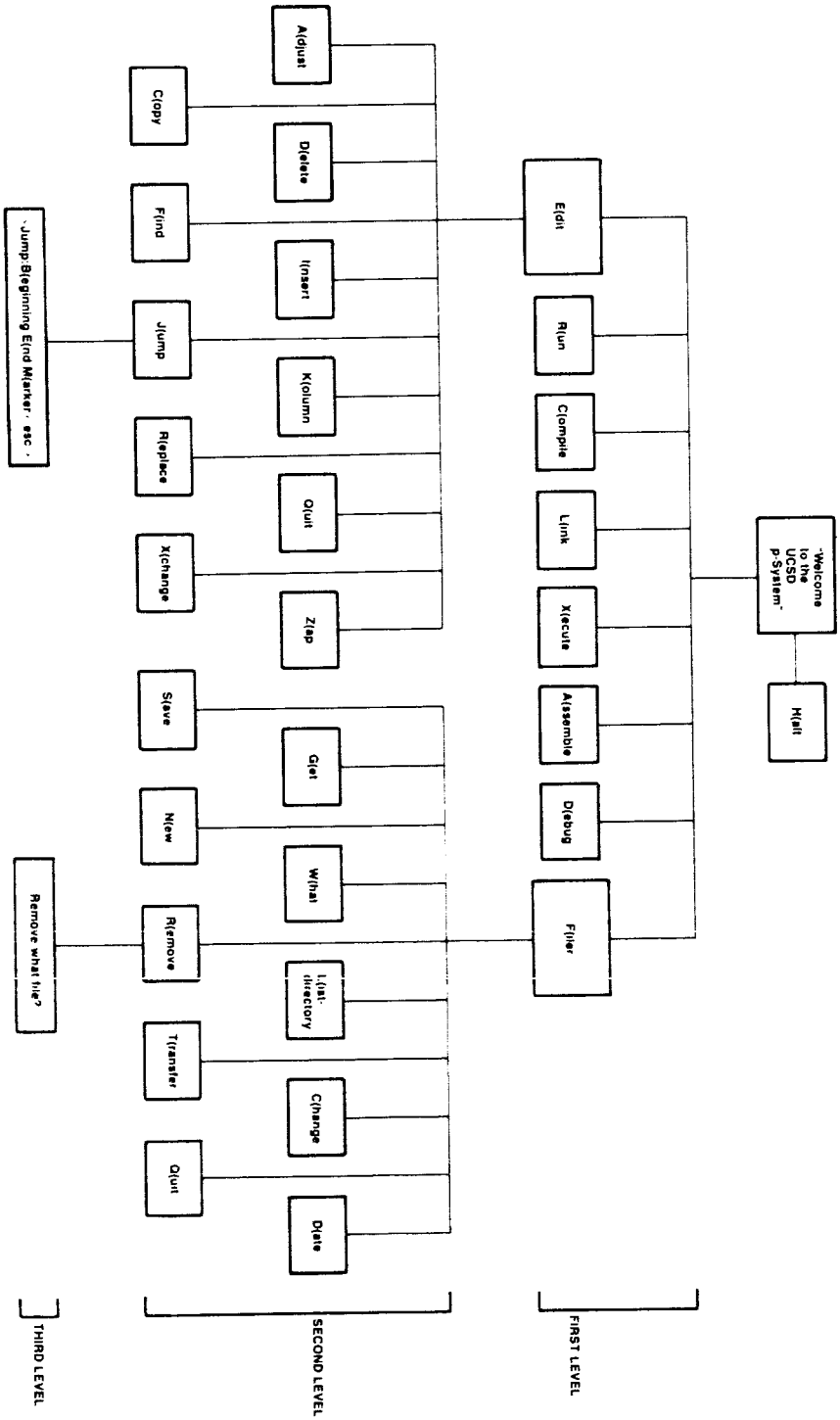

Figure 1-1. Command Hierarchy

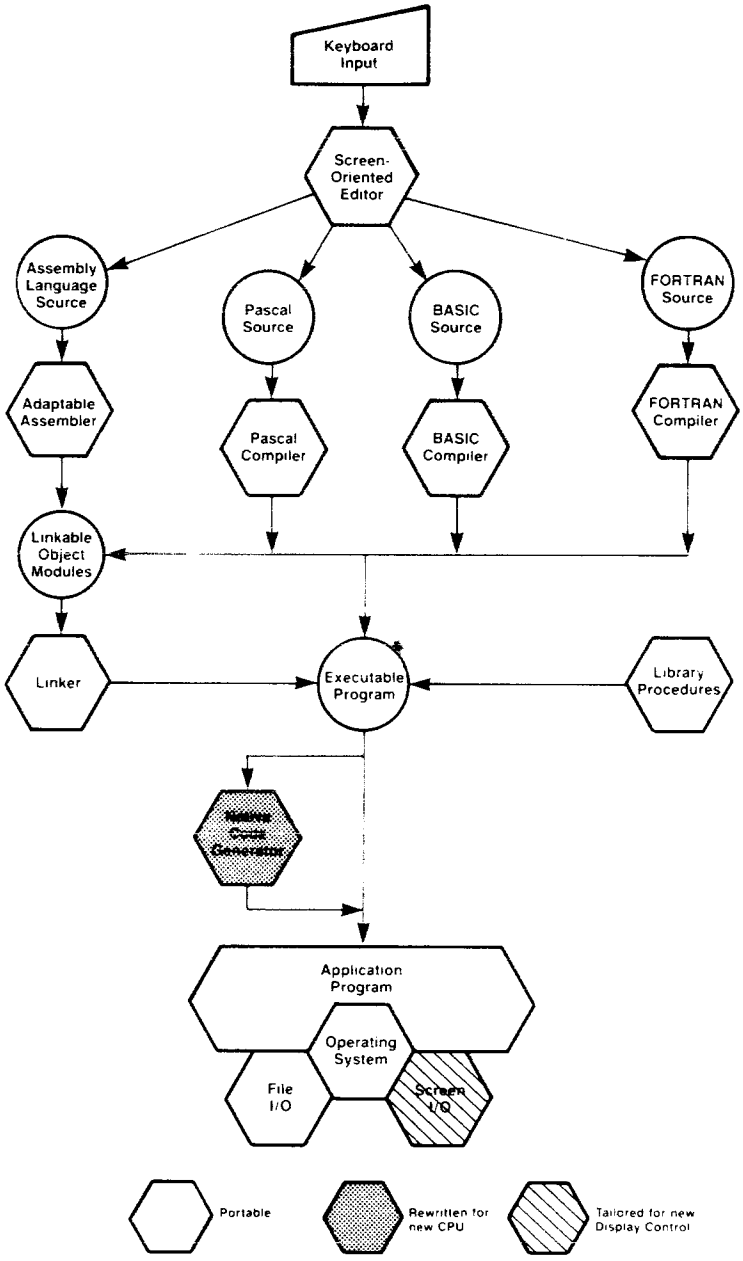

Figure 1-2. Major p-System Components

System Files

The system files are disk files which contain the bulk of the 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 don't 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 current system editor that you can call by pressing 'E' for E(ditor—as displayed on the Command menu. This may be the screen-oriented editor or any other editor that the p-System uses. To use another editor, simply change its file name to SYSTEM.EDITOR after changing the original SYSTEM.EDITOR to something else.

SYSTEM.COMPILER may contain a Pascal, FORTRAN, or BASIC compiler. To call the code files for FORTRAN or BASIC compilers, with a single keystroke from the Command menu, change the name of the desired files to SYSTEM.COMPILER.

SYSTEM.ASSMBLER is the adaptable assembler that translates assembly language into machine code. It is the assembler for a particular processor (there are several varieties available). You can change to a different processor by changing the name of the appropriate assembler to SYSTEM.ASSMBLER. The assemblers are p-code files and are, therefore, machine-independent. They can be used to assemble codes for processors other than the host proccessor. The assembler needs an opcodes and an errors file.

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 on the system disk when it is bootstrapped, 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 system disk, it is executed every time the Command menu is about to be displayed. This facility is generally used as a menu driver for turnkey applications. (If SYSTEM.STARTUP exists, it is executed before SYSTEM.MENU is called for the first time.)

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

p-SYSTEM CONFIGURATIONS

There are two main p-System configurations. The first configuration, which is standard, displays the Command menu when it is booted. As already mentioned, from this menu, you may select the major p-System components and execute programs.

The second configuration, on the other hand, never displays the main p-System menu. It is intended to be bundled with application programs which display their own menus and prompts. When this configuration is employed, the p-System is "hidden" underneath the application's own environment.

Operating System

CHAPTER 2

THE

OPERATING SYSTEM

INTRODUCTION

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 sections 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 ?
Filer: G(et, S(ave, W(hat, N(ew, L(dir, R(em, C(hng, T(rans, D(ate, ?
>Edit: A(djust C(opy D(el F(ind I(nsert J(ump K(ol M(argin P(age ?
```

If you enter '?' at the Command menu, the following is displayed:

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

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 commands by pressing 'Q' (for Q(uit). After performing a function, you may press <space> 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 'X'; 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 <return>.

If you make an error while entering your response, you can press the <backspace> key to correct it. You can also use <delete line> to erase your entire response. You can then resume entering the correct response.

Another example of a prompt is when you use the filer to list the directory of a volume. After pressing 'F' on the Command menu to display the F(iler menu, and then 'L' on the F(iler menu, the following prompt will be displayed:

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

Your response to this prompt would be to enter in any valid volume name and then to press <return>.

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 'A' to select the A(ssembler. The system then displays the following prompt:

```
Assemble what file?
```

Enter YOUR.FILE and press <return>. Assuming that YOUR.FILE.TEXT exists, the system displays the following prompt:

```
Code file name?
```

Enter '$' and press <return>.

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 asembled listing: (<CR>) for none)
```

You could enter PRINTER: and press <return>. 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 screen:

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

In the preceding example, the system couldn't find the disk VOLNAME and waits until you press <space>. (If you press <space> but haven't replaced VOLNAME, the system redisplays 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. Also, the editor is covered in Chapter 4.

The assembler and linker are covered in a separate assembler manual.

The compiler and debugger are covered in the Program Development Reference Manual.

A(ssemble

On the menu: A(ssem

This command starts the assembler SYSTEM.ASSMBLER (note that there is a missing "E"). If a work file is present, then *SYSTEM.WRK.TEXT or the designated file is assembled to a code file of a given machine code (depending on which of the assemblers has been named SYSTEM.ASSMBLER). 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 <return> for the source file, the assembly is aborted. Similarly, if you press <esc> followed by <return> for the code file or listing file, the assembler is existed.

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 *xxxx.ERRORS is present, where xxxx is the correct processor name, that is, Z80.ERRORS). It gives you some options:

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

You may continue the assembly by pressing <space>; abort the assembly by pressing <escape>; or, proceed directly to the editor to correct the source file by pressing 'E'. In the latter case, the system positions the cursor where the error was detected.

The assembler is described in a separate assembler manual.

C(ompile

On the menu: C(omp

This 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. If you press <return> for the code file, the default code is *SYSTEM.WRK.CODE. If you simply press <return> for the source file, the compilation is aborted; and, if you press <esc> <return> for the code file, the compilation is aborted.

Next, the compiler asks for a listing file. This may be a disk file (such as LIST.TEXT) or communications volume (such as PRINTER:). If you simply press <return>, no listing file is generated. If you press <esc> followed by <return>, 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 <space>, abort compilation by pressing <esc>, or proceed directly to the editor to correct the source file by pressing 'E'. 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 the Program Development Reference Manual and the UCSD Pascal Handbook. The FORTRAN and BASIC compilers are covered in separate reference manuals.

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 doesn't contain the debugger, you need to use the Library utility (described in the Program Development Reference Manual) 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 break point 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 p-machine architecture and must understand the p-code operators, stack usage, variable and parameter allocation, and so on. These topics are discussed in the Internal Architecture Reference Manual.

For more information about the symbolic debugger, refer to the Program Development Reference Manual.

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 Editors," 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 functions for managing files, manipulating work files, and maintaining disk directories. (Refer to Chapter 3, "File Management," in this manual, for detailed coverage of the filer.)

H(alt

On the menu: H(alt

This command stops System operation. To restart the p-System after a H(alt, you usually need to reboot it. Some systems may automatically reboot in response to this command.

On most single user personal computers, use of the H(alt command is optional. It is often sufficient to remove the system disks and turnoff 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 aren't 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 doesn't clear any I/O redirection, but 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 a separate assembler manual.

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 'M' to start the M(onitor. The system then displays the following menu.

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

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

All your input will be 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 monitor mode and saves the monitor file. If no monitor file is open, an error message is displayed. (You must use S(uspend or R(esume to return to the Command menu.)

A(bort terminates monitor mode but doesn't save the monitor file. (You must use S(uspend or R(esume to return to the Command menu.)

S(uspend turns off monitoring but doesn't 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 where you can add to it by using R(esume.

R(esume starts monitoring again and returns you to the Command menu. If monitoring wasn't 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 hasn't recorded special characters that the editor doesn't 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 can't 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 can't be found, the message:

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

is displayed. If the program requires assembled code which hasn't 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 equals sign (=), possibly followed by a file name or literal string.

The following table is a list of 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 the Program Development Reference Manual. 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, however.

The following items define the actual 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. Redirection from within your program may be accomplished through procedures in a unit called COMMANDIO. See the Program Development Reference Manual for more information.

Prefixes and Libraries

You can change the default prefix with the P= execution-option string. After this is done, all file names that don't 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 'X' to select X(ecute. Enter 'p=disk2' and press <return>. 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 the Program Development Reference Manual.)

To change the default library text file, press 'X', then 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= 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).

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

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).

I= 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= 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 pressed 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 exits 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 aren't used.

System redirection ends when the system terminates with a halt or a run-time error. An I(nitialize command doesn't 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, can't be redirected. Also, BLOCKREAD and BLOCKWRITE aren't redirected. This means that if you redirect a program which uses any of these operations, they won't be redirected.

Redirection also can't affect calls in the following form because these calls don't 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 'X' 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"
```

Filer

CHAPTER 3

FILE MANAGEMENT

INTRODUCTION

This chapter covers topics which are relevant to managing the files on your disks.

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 cards is covered. Wild cards 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 loose. 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 activities 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 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") have "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 functions that allow you to create, name and rename files, remove them, transfer them around, 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. Figure 3-1 illustrates the technical syntax for file names, and Figure 3-2 shows the syntax for volume names.

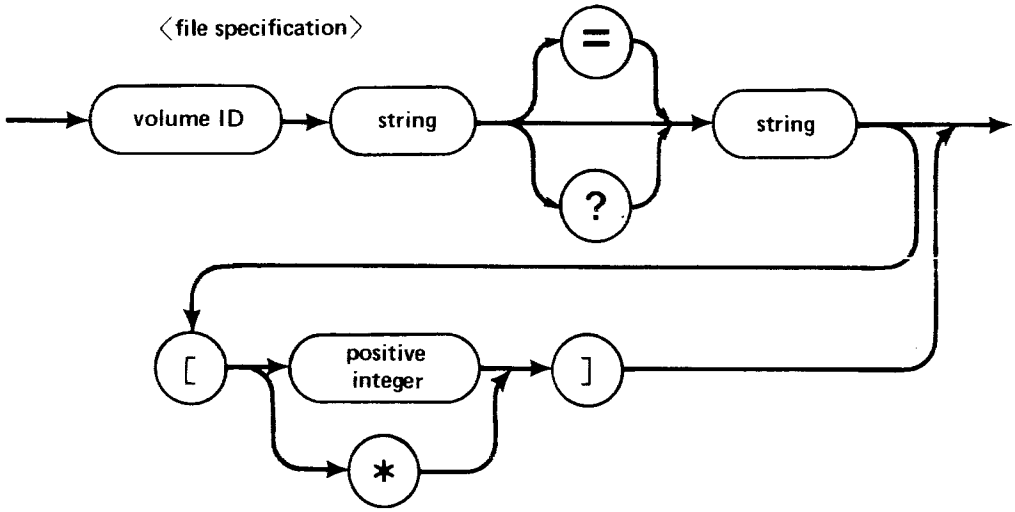

Figure 3-1. File Name Syntax

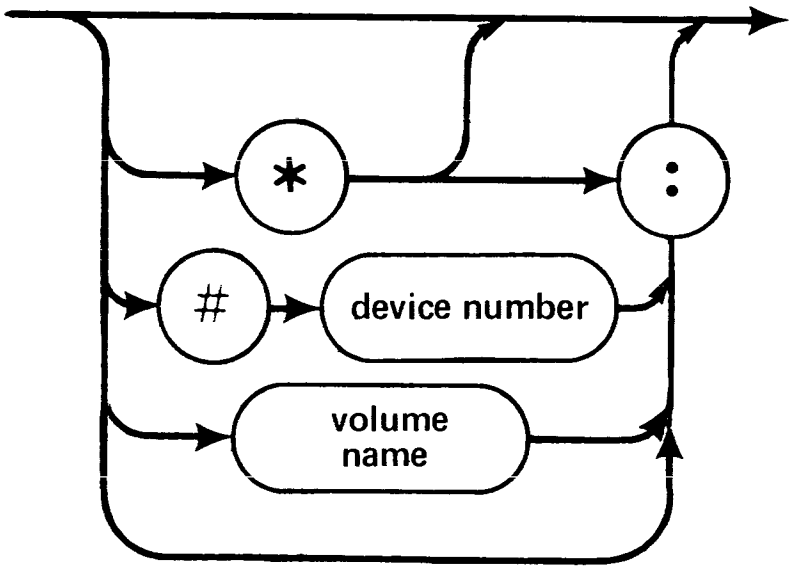

Figure 3-2. Volume ID Syntax

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 (/)
- Back slash (\ )
- 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, 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 #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 activity. 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 don't 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] 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, don't have any special suffix.

Here are some example file names which use these suffixes:

.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 (1024 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.

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

Each line of text is terminated with an ASCII CR. A line may begin with a blank-compression pair which consists 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 Internal Architecture Reference Manual for detailed description 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 functions (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) doesn't have internal structure; it simply produces or consumes a stream of data. For example, the printer and console are communication devices.

Appendix C illustrates the reserved volume names and device numbers used to reference the standard communication and storage devices.

The system distinguishes between storage and communication devices. Storage devices are usually disk drives. They contain 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 floppy 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 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.

WORK FILES

The work file is a scratch pad 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 and
*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, a compiled listing.

Whenever the editor alters a program contained in *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 'F' to enter the F(iler. 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?
```

Enter '?'. The system then displays more filer functions:

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

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

In the filer, answering a Yes/No question with any character other than 'Y' or 'y' constitutes a no answer. Pressing <esc> returns you to the main F(iler menu.

Many activities 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 can't find, the system displays the following message:

```
NAME: No such vol on-line <source>
```

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

NOTE: 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 function 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 a <return>. 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(ransfer) 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 Cards

Wild cards 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 matching the wild card specified. 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 specification must be of the form:

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

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

The following paragraphs describe the use of the filer with wild cards.

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
TEMP2              5  1-Jan-83
TEMP.CODE          2  1-Jan-83
```

With the Command menu displayed, press 'F' to call the F(iler. Then press 'R' to use the R(emove option. The system will display the following prompt:

```
Remove what file?
```

Enter 'TEMP=' press <return>.

The system then displays the following listing:

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

To verify and complete this operation, press 'Y'. To stop the operation, press 'N'. If you press 'N', the files won't be removed.

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

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

Enter '=TEXT' and press <return>. 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 wouldn't 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 'F' (Command menu) and then press 'T' to select the T(ransfer option. The system will display the following prompt:

```
Transfer what file?
```

Enter '#5:RE.USE.TEXT,*$' and press <return>. 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 won't 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 function is virtually certain to do this; avoid it.

With the Command menu displayed, press 'F' to call the F(iler and then press 'E' to use the E(xtended list function. The E(xtended list function will display the names of files in the directory and any unused blocks that may 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 F(iler menu displayed, press 'M' to use the M(ake function. 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 function, 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 function 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 function 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 function:

```
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 'M' (F(iler menu) to use the M(ake function and enter FILLER[75]. This procedure fills up the 75 blocks of unused space on the disk. Next, using the M(ake function, 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 function 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
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 doesn't 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 don't do this, the recreated file won't be executable.

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

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 the Program Development Reference Manual.)

Duplicate Directories

A duplicate directory can assist you in recovering from the situation where the main directory has been destroyed. 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. (A duplicate directory won't help you if you accidentally remove a file since the file is removed from both directories at the same time.)

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 function to do this when you are initializing a disk's directory. When the prompt 'Duplicate dir?' appears, press 'Y' for yes. This prompt also appears in the M(ake function 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 the 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 F(iler E(xtended list or L(ist functions 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 (aproximately 6) of MYDISK: to be overwritten with a listing of the directory of MYDISK:.

Response:

```
MYDISK:, DISK2:
```

This causes the directory of DISK2: to be overwritten.

In the latter case, you must use the disk recovery methods already desribed. In the first two cases, recovery isn't so difficult, even if there wasn't 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 the printer, like this:

```
Transfer what file? MYDISK:, PRINTER:
```

Generate hard copy of the directory and then use the filer to Z(ero MYDISK:. The Z(ero function won't alter the contents of MYDISK:, only the directory itself. Now use the M(ake function 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 function and the file name suffix, .SVOL. As with any other file the M(ake function 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]
```

An .SVOL file must be made at least 11 blocks long.

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

```
Zero subsidiary volume directory?
```

If you respond with a 'Y', the directory of the new subsidiary volume is zeroed. If you press an 'N', the directory isn't 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 isn't displayed, then there wasn't 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 with 'Y' if you want a duplicate directory to be maintained on the subsidiary volume, and 'N' 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 mounted unless the maximum number of subsidiary volumes has already been mounted. (Mounting and dismounting of subsidiary volumes is described in the next section.) You may then access and use it like any other p-System volume. The filer function, 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 activities 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 will overwrite 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 function. From the main F(iler menu, press 'O'. The system will display the following menu:

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

Press 'M'. The system display this prompt:

```
Mount what vol ?
```

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

```
Dismount what vol ?
```

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 volume? 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.

The other item on the O(n/off-line menu is C(lear. When this is selected, all subsidiary volumes are dismounted.

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 the Adaptable System Installation Manual).

NOTE: If you C(hange either the name of a subsidiary volume or the name of the corresponding .SVOL file, it is a good idea to 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:
```

If you don't do this, the .SVOL file and its corresponding subsidiary volume won't have the same name which might be confusing.

NOTE: If you want to T(ransfer one subsidiary volume to another, use the file-by-file method:

```
Transfer what file? SVOL1:=
To where? SVOL2:$
```

It isn't a good idea to do a volume-to-volume T(ransfer.

NOTE: If you need to extend the size of a subsidiary volume, do not use the DISKSIZE utility. You should M(ake another subsidiary volume the size you want and transfer the files from the old subsidiary volume to the new one.

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.

USER-DEFINED SERIAL DEVICES

The user-defined serial device facility allows you to take advantage of special serial I/O hardware capabilities. You can use this facility, along with the standard serial I/O devices (CONSOLE:, REMIN:, and REMOUT:), on some computers.

You may have up to 16 user-defined serial devices, in addition to a printer, a console and a remote line. User-defined serial devices may include additional printers, additional consoles, communication lines between users in a multi-user environment, and so on.

This feature isn't available with the adaptable system BIOS.

You can use SETUP, described in Chapter 5, to specify the number of user-defined serial devices that you will have.

FILER FUNCTIONS

This section describes filer functions and gives examples of their use. Functions are listed in alphabetical order with each new function beginning on a new page.

B(ad Blocks

On the menu: B(ad-blks

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

This function 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 320 blocks ? <y/n>
```

Enter 'Y' for yes to scan for the entire length of the disk. To check a smaller portion of the disk, press 'N'. 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 function changes file or volume names.

C(hange requires two 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 a <return> or a comma (,). 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:

```
C(hange 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 function doesn't 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 specifications are legal in the C(hange function. If you use a wild card character in the first file specification, then you must use a wild card 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 won't 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:

```
DISK1:MA=TEXT<return>
```

Prompt:

```
Change to what?

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 function 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 1-Jan-83
New date?
```

You may enter the correct date in the format given. After pressing <return>, the new date is displayed. Pressing only a return doesn't 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<return>'; 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 function.

NOTE: Some p-System application program are designed to examine and/or change the system date, either from your input, as with the filer, or automatically from battery operated clocks which are available with some machines.

E(xtended List

On the menu: E(xt-dir

This function lists the directory in more detail than the L(dir function. (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 options and prompts are used in the same way as the L(dir function.

Since this function shows the complete layout of files and unused space on the disk, it is useful in conjunction with the M(ake function. (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 <space> to list the rest of the directory. Press <esc> 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, 356 unused, 200 in largest
```

F(lip Swap/Lock

On the menu: F(lip swap/lock

This function 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 unneccessary 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 isn't 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 two-drive systems. It is 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; and in the second mode, they are memlocked. The F(lip swap/lock function allows you to choose the mode the filer will use. Upon entering the filer, the initial state is always the memswapped state. Pressing 'F' acts as a toggle between the memswapped and memlocked states.

For example, if you enter the filer and press 'F' 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 doesn't have enough space to memlock the filer segments, you receive a message indicating so. (If there aren't at least 1500 extra words available, the filer won't allow the memlock option.)

G(et

On the menu: G(et

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

The entire file specification isn't necessary. If the volume ID isn't given, the default disk is assumed. Wild cards aren't 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 function. If you reboot the p-System before using the S(ave function, the p-System forgets the name of the work file. In this case, the p-System doesn't know what files the SYSTEM.WRK files were derived from.

K(runch

On the menu: K(rnch

This function 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 hasn't 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 function before using the K(runch function unless all files are also backed up on a different volume.

If the K(runch function must move SYSTEM.PASCAL or SYSTEM.FILER on the system disk, it then displays a prompt which asks you to reboot 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 320 ? (y/n)
```

The "320" indicates the last block on your volume and may be different for your disks. To start the K(runch, from this location, press 'Y'. To start the K(runch at another location, press 'N' and this is displayed:

```
Starting at block # ?
```

Enter the block number at which the K(runch should begin.

The contents of subsidiary volumes can be K(runched just like any other volume.

L(ist Directory

On the menu: L(dir

This function 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 device, 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 expression for multiple file names. In this case, the single file or the subset of the directory indicated by the wild card 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, enter 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 <space> causes the rest of the directory to be listed; pressing <esc> 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:

```
Dir listing of what vol?

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 cards:

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.

Prompt:

```
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 function creates a directory entry with the specified file name.

M(ake requires you to enter a file name. Wild card characters aren't 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 under "File and Volume Names" earlier.

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 initialized 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 informaton about this, see the section, "Subsidiary Volumes."

N(ew

On the menu: N(ew

This function clears the work file.

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

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

Entering 'Y' clears the work file, while 'N' 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 function mounts or dismounts subsidiary volumes.

With the filer menu displayed, press 'O'. The system displays the following menu:

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

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

```
Mount what vol ?
```

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

```
Dismount what vol ?
```

To dismount all the subsidiary volumes, press 'C'. 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 about subsidiary volumes, see the subsidiary volume section earlier in this chapter.

P(refix

On the menu: P(refix

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

This function 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 prefix 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 (:) 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 will be 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 function terminates the filer and returns you to the Command menu.

R(emove

On the menu: R(em

This function 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 function; not R(emove. Using R(emove may confuse the system.

Before finalizing any removals, the filer displays the following prompt:

Prompt:

```
Update directory?
```

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

As noted before, wild cards in R(emove activities are legal.

EXAMPLE:

Prompt:

```
Remove what file?
```

Response:

```
=CODE
```

Causes the filer to remove RUNIT.CODE and COPYIT.CODE.

Pressing the wild card question mark (?) causes the R(emove function 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.

WARNING: 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 function saves the work file under the file name you specify.

The entire file specification isn't necessary. If the volume ID isn't given, the default disk is assumed. Wild cards aren't allowed, and the size specification option is ignored.

EXAMPLE:

Prompt:

```
Save as VOLNAME: FILENAME?
```

The first prompt appears if your work file was derived from an existing file. It asks you if you want to save it under the old file name. Press 'Y' if you do, and 'N' otherwise.

The second prompt appears if your work file was created from scratch, or if you respond 'N' to the first prompt.

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.

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 function 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 cards are permitted in file name specifications only. Size specification information is recognized for the destination file. If you include a size specification, the file is placed in the first unused area on the disk which is at least as large as the size specification indicates.

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, don't 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 <space>.

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:$
```

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

For example, if in response to the T(ransfer function prompt, 'Transfer what file', you respond with MYDISK:DOCU.TEXT,BACKUP:, the system will display the following prompt.

```
Destroy BACKUP: ?
```

A 'Y' answer causes the directory of BACKUP: to be destroyed.

NOTE: If the file to be transferred is two blocks long or less, the system won't display the warning prompt. The file is transferred to the area where the bootstrap normally resides (in front of the disk's directory).

You may transfer files to volumes that aren't storage volumes, such as CONSOLE: and PRINTER:, by specifying the appropriate volume ID (see Appendix A) in the destination file specification. Don't 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 won't know when to stop transferring. File names accompanying a communication device are ignored.

Wild cards are recognized in the T(ransfer function. 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 tne destination file specification (replacement strings). Any of the subset-specifying or replacement strings may be empty. The filer considers the file specifications equal sign (=) or question mark (?) 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 don't 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, the filer rewrites the file to the size-specified area and removes the older copy—without changing the file's size.

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: Some disks have areas which aren't accessible by the system. The filer can't transfer those areas. Bootstraps, in particular, may have to be transferred with the utility BOOTER. See the Adaptable System Installation Manual.

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: ?
```

WARNING: 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 function to change the name of the backup disk. The two disks shouldn't 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 function 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: [12000]
  5 # FLOPPY1: [  320]
  6   PRINTER:
 12 # 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 12000 blocks of storage capacity, and the floppies on-line in drives #5: and #12: contain 320 and 640 blocks, respectively.

The V(olumes function 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: [12000]
  5 # FLOPPY1: [  320]
  6   PRINTER:
 12 # 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 3000 blocks.

W(hat

On the menu: W(hat

This function identifies the name of the current work file. If the work file hasn't been saved, the phrase "(not saved" is displayed after the file name.

EXAMPLE:

```
Work file is DOC1:STUFF
```

X(amine

On the menu: X(amine

This function 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>
```

Prompt:

```
Block-range ?
```

Response:

```
<block-number>
      or
<block-number> — <block-number>
```

If you just enter a block number, only that block is examined. If you enter two numbers separated by a hyphen, all of the blocks from the first one to the second one, inclusive, are examined. 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?
```

Entering 'Y' causes the filer to examine the blocks and return either of the messages:

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

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 aren't 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.

WARNING: A block that is fixed may contain garbage. "May be ok" should be translated as "is 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 function 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> ?
```

A 'Y' response generates...

Prompt:

```
Duplicate dir ?
```

If you enter a 'Y', 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 function was used:

Prompt:

```
Are there 320 blks on the disk ? (y/n)
```

'Y' accepts that number of blocks and skips the next prompt. 'N' generates...

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 ?
```

'Y' accepts the name. 'N' returns to the prompt requesting a new volume name. If the filer succeeds in writing the new directory on the disk, this message is displayed:

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

Editor

CHAPTER 4

SCREEN-ORIENTED EDITOR

INTRODUCTION

The editors available with the p-System allow you to create, alter, and examine text files. Text files contain human-readable material such as memos or manuscripts.

Three editors are available with the p-System: the Screen-Oriented Editor, the advanced editor (EDVANCE), and the Line-Oriented Editor (YALOE). This chapter is devoted to the Screen-Oriented Editor.

THE EDITOR

Introduction

In order to use the editor, SYSTEM.EDITOR must reside on a disk which is on-line. Also, the SYSTEM.MISCINFO file must be configured for your particular terminal. If this hasn't 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 screen 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 screen. When an editor command takes you to a position in the file that isn't presently displayed, the window moves to show you that new portion of the file.

The Cursor

The cursor is usually a small rectangular box or an underline that appears to be on (or under) a character. On some terminals, the cursor may blink continuously. The cursor is logically located between the character to its left and the character on which it rests. You position the cursor to indicate to the editor its commands are to affect the text. For example, the editor will insert text in front of (that is, to the left of) the charcter 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 screen 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 E(dit menu.

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

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 'E' from the Command menu. If you don't have a text work file, you are prompted for the name of a file to edit. You should enter the file name without the ".TEXT" suffix, followed by <return>. (If you have a text work file, that file is automatically edited.) The system will display the E(dit menu:

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

The E(dit menu is the second level of the command hierarchy, as is the F(iler menu and all the other menus that you can display from the Command menu.

For example, to select the editor I(nsert option, press 'I'. 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, enter '2R' 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 editor will perform the specified function two times.

If you don't 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 will move to the right (toward the end of the file) when you press the space bar. If the direction indicator is reversed, then the cursor will move left (toward the beginning of the file) when you press <space>.

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.

Using the Editor

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 doesn't affect the arrow keys and <backspace>.

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 isn't 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 equal sign 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 won't 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 screen window forward (or backward) by one screen and positions the cursor to the beginning of the line. These commands are described in the section entitled, "Screen-Oriented Editor Commands."

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>	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.

To move the cursor on terminals which don't have arrow keys, use the SETUP utility to designate a set of control keys to act as cursor keys. To configure the system for use with a particular terminal, refer to Chapter 5.

You can't move the cursor outside the text of the program. For example, after the 'N' in 'BEGIN' in Figure 4-1, press the <right-arrow>; this moves the cursor to the 'W' in 'WRITE'. Similarly, at the first 'W' in "WRITE('TOO WISE ');", use the <left-arrow> to back up after the 'N' in 'BEGIN'.

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

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

Figure 4-1. Cursor Example

In Figure 4-2, 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 <down-arrow> twice. To mark the positions the cursor occupies, labels a, b, and c are used in Figure 4-2. The 'a' marks the initial position of the cursor; the 'b' marks the cursor position after the first <down-arrow>; and the 'c', marks the cursor after the second <down-arrow>.

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

Figure 4-2. Cursor Positions

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

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 isn't 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 'S'. 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>. Entering '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, use the S(et E(nvironment command.

Work Files

When you enter the editor, the system reads and displays the work file. If you haven't 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 Figure 4-3. After entering the name, the text of the first part of the file appears on the screen.

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

Figure 4-3. Program String1

2. With a <return>. This response indicates that you wish to start a new file. The only thing visible on the screen after this response is the E(dit menu. Press 'I' to begin inserting a program or text.

3. With <escape>. This response stops the editor, causing the system to return to the Command menu.

Using Insert

To use the I(nsert command, press 'I' from the E(dit 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 'I'. From the point of insertion, the rest of the line is moved toward the right side of the screen. If the insertion is long, that part of the line is moved down to allow room on the screen.

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

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

The cursor is at the 'W' in 'WISE' (see Figure 4-3). Enter 'SMART'. The word appears on the screen as it is entered (see Figure 4-4).

The choice at the end of the prompt indicates that pushing the <etx> key accepts the insertion; while pushing the <esc> key rejects the insertion, leaving the text as it was before pressing 'I'. Press <etx> (see Figure 4-5).

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

Figure 4-4. Screen after entering 'SMART'

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

Figure 4-5. Screen after <etx>

While in I(nsert, you can insert a carriage return by pressing <return>. 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. (See the section on Auto-Indent mode.)

Using Delete

D(elete works like I(nsert. Move the cursor to the 'W' IN WISE (see Figure 4-5) and press 'D' to select the D(elete command. The system then displays the following prompt:

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

Each time you press <space>, a letter disappears from the screen. Press <space> four times. Pressing <backspace> causes a character to reappear. Pressing <etx> causes the deleted text to be removed permanently, or pressing <esc> causes it to reappear and remain unaffected.

To delete a carriage return at the end of a line, press 'D' and then press <space> until the cursor moves to the beginning of the next line.

Leaving the Editor

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

```
>Quit:
U(pdate the work file 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 isn't necessarily your work file.

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

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

Screen-Oriented Editor Commands

The Screen-Oriented Editor activities 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 'A' from the E(dit menu. This displays the following menu:

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

The A(djust command moves a line to the left or to the right. The <right-arrow> and <left-arrow> move the line on which the cursor is located. Each time you press a <right-arrow>, the whole line moves one space to the right. The <left-arrow> moves the line one space to the left.

To adjust more than one line, use the <up-arrow> or <down-arrow>; the line above or below the previously adjusted line is automatically adjusted by the same amount.

The character 'L' justifies the line to the left margin, 'R' justifies it to the right margin, and 'C' centers the line between the margins. Use the <up-arrow> and the <down-arrow> 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 <etx> to exit the A(djust command; <esc> won't work here.

C(opy

On the menu: C(opy

Repeat factors are not allowed.

Press 'C' from the E(dit menu. The following menu is displayed.

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

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," or a text file on disk. To copy from the copy buffer, press 'B'. 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 is placed at the end of the copied text.

The following commands affect the copy buffer.

1. D(elete: When you press <etx>, the buffer is loaded with the deletion. When you press <esc>, the buffer is loaded with what would have been deleted.

2. I(nsert: When you press <etx>, the buffer is loaded with the insertion. When you press <esc>, 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.

Generally, if the text that you want to copy already exists, you should D(elete it, and press <esc>. Then you can use C(opy B(uffer to place that text anywhere you like. The original text remains unaffected.

To copy text from another file, press 'F'. 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 aren't allowed.

To select the D(elete command, press 'D' from the E(dit menu. The following prompt is displayed:

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

You must have first placed the cursor where you want to begin deleting text. The D(elete command 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 <etx>; to escape, press <esc>.

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

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?
```

A 'Y' or 'y' is a yes answer; any other character escapes the D(elete command.

The following procedure shows how to use the D(elete command (see Figure 4-6).

1. Move the cursor to the 'E' in END.

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

3. Press 'D'.

4. Press <return><return>. After pressing <return> 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 <etx>. After deletion, the program appears as shown in Figure 4-7.

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.

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

Figure 4-6. D(elete Example A

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

Figure 4-7. D(elete Example B

F(ind

On the menu: F(ind

Repeat factors are allowed.

To use the F(ind command, press 'F' from the E(dit 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 'F'; this number is 1 if you gave no repeat factor.)

The F(ind command 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 doesn't 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 command.

In the STRING1 program (see Figure 4-8), with the cursor at the first 'P' in 'PROGRAM STRING1', press 'F'. When the prompt appears, enter 'WRITE'. Single quote marks must be entered. The prompt with your response is shown in the following listing.

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

Immediately, the cursor jumps to the character following the 'E' in the first 'WRITE'.

In the STRING1 program with the cursor on the 'E' in 'END.', enter '<3F' (don't include single quotes). This entry finds the third occurrence of the pattern in the reverse direction. When the menu appears, enter '/WRITELN/'. The menu with your 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.

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

Figure 4-8. F(ind Example

Insert

On the menu: I(nsert

Repeat factors aren't allowed.

To select the I(nsert command, press 'I' from the E(dit 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 character <return>. Nonprinting characters are echoed with the nonprinting character symbol (usually a '?'; this can be changed by using SETUP). To make corrections while still in I(nsert, use <backspace> (<bs>) to remove one character at a time or <rubout> (<del>) 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 auto-indent and the filling modes.

Using Auto-Indent

If auto-indent is true, a <return> causes the cursor to start the next line with an indentation equal to the indentation of the line above it. If auto-indent is false, a <return> returns the cursor to the first position of the next line.

Using Filling

If filling is true, 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 two returns in succession creates a new paragraph. In other words, a paragraph is a block of text delimited by blank lines (or command lines (see S(et), or the beginning or end of the text file). The first line of a paragraph may be indented differently than the remaining text (see S(et E(nvironment).

If both auto-indent and filling are true, auto-indent controls the left-margin, while filling controls the right-margin. You may change the level of indentation by using the <space> and <backspace> keys immediately after a <return>.

Example 1: With auto-indent true, the following sequence creates the indentation shown in Figure 4-9.

```
'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
```

Figure 4-9. Indentation Example

Example 2: With filling true (and auto-indent false) the following sequence creates the indentation shown in Figure 4-10.

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


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

Figure 4-10. Auto-Indent Example

You can force the cursor to the left margin of the screen by entering <control-Q> (ASCII DC1). On some machines or terminals, CTRL-Q is the prefix character which requires you to press it twice to achieve the desired effect.

Filling also causes the editor to adjust the margins on the portion of the paragraph following the insertion. This adjustment doesn't 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 Auto-indent 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 doesn't 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 <esc> is used, there is no string available for C(opy.

J(ump

On the menu: J(ump

Repeat factors aren't allowed.

Upon entering J(ump, the following menu appears:

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

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

```
Jump to what marker?
```

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

K(olumn

On the menu: K(ol

Repeat factors aren't allowed.

K(olumn displays the following menu:

```
>K(olumn: <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 <left-arrow> or to the right by using the <right-arrow>. Using the <up-arrow> or <down-arrow> applies the same column adjustment to the line above or below. Press <etx> 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 <left-arrow> deletes one character at the cursor. It's easy to do this and any characters deleted aren't saved in the copy buffer as in D(elete, so be careful when using K(olumn.

M(argin

On the menu: M(argin

Repeat factors aren't 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 'M'.

Figures 4-11 and 4-12 show margins settings and an example of a paragraph that uses 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.
```

Figure 4-11. M(argin Example A

```
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.
```

Figure 4-12. M(argin Example B

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.

Command Characters

M(argin won't 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 in the global direction. If a repeat factor is used, several screens are traversed. The cursor remains on the same line on the screen, but is moved to the start of the line.

Q(uit

On the menu: Q(uit

Repeat factors aren't allowed.

Q(uit displays the following menu:

```
>Quit:
U(pdate the work file 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 aren't recorded on disk. All editing during the session is irrecoverably lost, unless you have already used the W(rite command 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 command puts 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 <return> 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 V(erify command ('V(fy') allows you to examine each <targ> string found in the text so you can decide if it is to be replaced. To use this command, press 'V' 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 'R' at this point causes the replacement to take place, and the next target to be sought. Pressing <space> causes the next occurrence of the target to be sought. At any point, an <esc> 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 can't 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' isn't 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( ',');
```

Enter '2R' from the E(dit 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 aren't allowed.

Upon entering S(et, the following menu appears:

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

S(et E(nvironment

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

```
>Environment:  {options} <spacebar> to leave
    A(uto indent  True
    F(illing      False
    L(eft margin  9
    R(ight margin 70
    P(ara margin  9
    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.

E(nvironment Options

A(uto indent:

Auto-indent affects only insertions. Refer to the section on I(nsert. Auto-indent is set to true (turned on) by entering 'AT' and to false (turned off) by entering 'AF'.

F(illing:

Filling affects I(nsert and M(argin. (Refer to those sections.) Filling is set to true (turned on) by entering 'FT' and to false by entering 'FF'.

L(eft margin, R(ight margin, P(ara margin:

When Filling is true, the margins set in E(nvironment are the margins that affect I(nsert and M(argin. They also affect the Center and justifying commands in A(djust. To set a margin, press 'L', 'R', or 'P', 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----T----T----
Column#1
```

The cursor will start at position one in the line of Ts and dashes (-). The line 'Column#1' indicates the positon 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 'C' and enter the desired column number. Press 'T' to insert a tab or delete a tab.

Pressing 'T' changes the indicator from a dash to T; pressing 'T' 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 true by entering 'TT' and to false by entering 'TF'. If Token is true, Token is the default; and if Token is false, Literal is the default.

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 command 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 followed by a <return>. 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 "a" through "t"; that space will now be available for use in setting the desired marker.

V(erify

On the menu: V(erify

Repeat factors aren't allowed.

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

X(change

On the menu: X(change

Repeat factors aren't allowed.

Upon entering X(change, the following menu appears:

```
>eXchange: TEXT {<bs> a char} [<esc> escapes; <ext> accepts]
```

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

For example, in the file in Figure 4-13, 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 Figure 4-14, with the cursor before the second 'S'.

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

Figure 4-13. X(change Example A

```
WRITE('TOO SMSE ');
```

Figure 4-14. X(change Example B

The <etx> key accepts the actions of the eX(change, while the <esc> key leaves the command with no changes recorded in only 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> to move the cursor about the screen. 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 the Adpatable System Installation Manual).

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 isn't enough room in the copy buffer. If this is the case, the editor then asks if you want to Z(ap anyway.

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

Utilities

CHAPTER 5

UTILITY PROGRAMS

INTRODUCTION

This chapter covers several utility programs that will help you use the p-System. 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.
- Making programs execute more quickly.
- Debugging programs.
- Showing you the internal details of files.

The utilities described in this chapter fall into the first four categories. The Program Development Reference Manual describes several utilities which fit in the last two categories. The Adaptable System Installation Manual also describes several utilities.

PRINT

Introduction

The PRINT utility provides a simple way for p-System users to print text files. The screen-oriented editors in the p-System make 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 document.

PRINT complements the other two principal mechanisms within the p-System for printing text files (the T(ransfer operation in the F(iler and the Print Spooler). Neither of those mechanisms provides any formatting support (such as inserting page breaks). 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 control features they may have, and can be used with either continuous forms or manual single sheet loading.

The following section describes the simplest uses of PRINT. You may never need to know more. If you do, read the rest of this section, 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); and

- That your printer advances the paper to a new page when the p-System sends a 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):

1. Enter the name of the file to be printed, using the I(nput option on the menu.

2. 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.

3. If you need to cause a page advance on the printer to tear off the printout(s) you've made, A(dvance should do the trick.

4. Finally, when you are done with a printing session, use Q(uit to leave PRINT.

If the printouts produced by this process aren't what you'd like, or if some of the assumptions above don't apply in your situation, read the rest of this section to discover how PRINT can be configured to serve your needs better.

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 options. They may:

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

2. Prompt you to enter a sequence of characters, followed by a <return>. These characters are a file name, in the case of I(nput.

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

4. 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.

There is also the '?' option which displays information about how to use the PRINT utility.

Other than the principal menu of commands, which occupies most of your display screen, PRINT does most of its communication with you through the top line of the screen. Once you have selected an option, prompts appear on this line to direct you. Error messages are also shown on this line and are usually left there until you press <space> 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 of 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 doesn't 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 starts with the command line flag character, it isn't printed. Usually, this means it is a COMMAND line that gives directions to PRINT. 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. If they don't, the line is simply ignored. (You can place comments in your text using this mechanism.)

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 '.INCREMENT' are both treated as include commands.) Commands may be in either uppercase 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 can't 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'
```

COMMAND

This command has one parameter, a single character (which still must 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 '!'
```

END

This command has no parameters. It indicates that no more text should be taken from the current file. If the current file is an included file, PRINT returns to the principal file. If the current file is the principal file, printing is discontinued.

END is convenient for EDVANCE users. The EDVANCE editor allows you to define special function key macros by using text inside of the file itself. Also, EDVANCE allows you to keep an automatic log of update information within a text file. Furthermore, whether or not you used EDVANCE, you may wish to have an area within your files where you keep miscellaneous informaton that you don't want to be printed along with the main portion of the file. Any of this sort of material can be placed after an END command. PRINT will ignore it. For example:

```
.end
Here you could have some
special key definitions for
EDVANCE.
```

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 sequences 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 which is 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 which 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're 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 6 lines per inch and you're 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 to mean "advance the paper to the next page." If your printer can't turn off the U(se form feed option, 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 Invocation 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 option has been included to address this situation.

This option 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 PRINT 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 examined 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 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 <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 created.

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 each line to be preceded by its sequence for the current page.
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 the following command, you can:

INCLUDE	Insert an additional file into the document being printed in place of the include command.
PAGE	Cause an immediate page break.
HEADING	Specify the contents of the heading for subsequent pages.
COMMAND	Change the command line flag character.
ESCAPE	Change the escape sequence flag character.
END	Terminate printing the current text file.

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.

PRINT SPOOLER

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 on-line 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 define the menu options:

P(rint Prompts for the name of a file to be printed. This name is then added to the queue. If SYSTEM.SPOOLER doesn't already exist, it is created. In the simplest case, P(rint may be used to send a single file to the printer. Up to 21 files may 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 (that is, 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 may run concurrently with spooled 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 p-System. The print spooler automatically restarts at boot time if *SYSTEM.SPOOLER isn't 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 menu, enter *SPOOLER.CODE, and press <return>. Then press R(esume.

QUICKSTART

Introduction

The QUICKSTART utility can be used to make programs start more quickly. A program's startup time is the amount of elapsed time between the moment the invocation of the program is requested and the moment the execution of the program actually commences. During this startup time the p-System is building the execution environment for the program.

A program's execution environment is a network of p-System data structures, together with the areas of memory required by the program for its data. Each compilation unit contained within the program has a table in the execution environment which it uses during its execution to refer to other compilation units.

The QUICKSTART utility constructs a description of the execution environment for a program and generates a code file for the program which contains this execution environment description. The operating system detects the presence of execution environment descriptions within program code files and attempts to reconstruct the required execution environment from such descriptions when the programs are invoked. For large programs built out of a collection of separately compiled p-System units, this reconstruction process is considerably faster than the normal execution environment construction process.

After QUICKSTART has been used on a code file, that code file may be invoked with the X(ecute command as usual.

The reduction in invocation time for a quickstarted program is achieved by reconstructing the program's execution environment from the description in the code file instead of building the environment from scratch each time the program is invoked. Except for the difference in invovation time, the execution of a quickstarted program is identical to that of the original program.

When a quickstarted program is executed, the system first inspects the program code file to determine if an execution environment description is present within the program code to reconstruct the execution environment required by the program from the description in the code file. If the code file doesn't contain an environment description, or the environment description contained within the code file is obsolete, the system attempts to build the environment for the program in the usual fashion.

QUICKSTART Utility Operation

This section describes the operation of the QUICKSTART utility program.

System Environment Preparation

As the first step in using the QUICKSTART utility, you must set up the system environment for normal execution of the program. This includes making sure that the proper volumes are on-line and that any required library files are available. Note that the QUICKSTART utility uses the same components for locating the units which are referenced by the program it is processing.

QUICKSTART provides a set of toggle options that control the manner in which the quickstarting of a program is accomplished. The settings of these options can influence the way in which you set up the system environment prior to running QUICKSTART. These toggle options are discussed next.

C(opy Toggle Option

The C(opy toggle option determines whether the output of QUICKSTART is a modified version of the original code file, or a new code file.

In its default setting, the C(opy toggle option is off. This causes QUICKSTART to modify the original code file. The new execution environment description is either appended to the end of the code file, or will be written on top of an old description already present. Using QUICKSTART in this manner avoids the rather slow process of making a copy of the original file; however, there is a chance that the insertion of the new execution environment description will fail due to insufficient disk space at the end of the code file. You should make sure that a section of unused disk space follows the code file. The number of unused blocks which are required depends on the size and complexity of the program.

When the C(opy toggle option is on, a new code file is created by QUICKSTART and the execution environment is appended to the end of that file. Any previous environment description embedded in the code file is discarded. This method of using QUICKSTART is somewhat slower. But it is safe and more likely to insure that the size of the code file can be extended if necessary to install the new environment description. In order to use QUICKSTART in this manner, there must be enough disk space for a copy of the entire code file on one of the on- line volumes.

L(ibrary Copy Toggle Option

QUICKSTART installs a checksum part number into the library code files which are used by the program. The checksum is utilized to detect when an environment description has become obsolete due to a change in one of those library code files. A new checksum is only inserted into a library code file if that file lacks a valid checksum. Because of this, it must be possible for QUICKSTART to write to the volumes containing library code files without valid checksums.

With some p-System installations, a referenced library code file may reside on a RAM disk rather than a physical disk. When QUICKSTART updates such a code file, the updated information will be lost the next time the computer is powered off. As an aid to users who use RAM disk, the QUICKSTART utility has another toggle option called L(ibrary copy. When the L(ibrary copy toggle option is on, QUICKSTART first updates the original copy of a referenced library code file with a new checksum, and then asks you if the updated library code file contents should be copied to another file. Thus this facility can be used to save the updated library code files on a physical disk.

M(essages Toggle Option

The QUICKSTART utility has the capability of writing detailed progress messages to the console. These progress messages provide you with the names and locations of the compilation units which are being included within the execution environment for the program being quickstarted. In addition, these messages advise you of the copying or modification of code files. Most of the time you won't require the large amount of information provided by the progress messages. The information can be useful, however, when you are trying to diagnose the cause of a malfunction in a program. The QUICKSTART utility has a M(essages toggle which controls whether or not progress messages are displayed. The default setting of the M(essages toggle option is OFF, which results in the progress messages being suppressed.

Using The QUICKSTART Utility

The QUICKSTART utility (QUICKSTART.CODE), displays this menu:

```
Quickstart: P(rogram, S(ystem, C(opy, L(ibrary, M(essages, Q(uit
Toggle settings:  Copy OFF, Library copy OFF, Messages OFF
```

The first line shows the set of commands recognized by QUICKSTART. The second line displays the current settings of the toggle options. The toggle option settings shown above are the default settings.

The C(opy, L(ibrary, and M(essages commands cause the setting of the corresponding toggle option to be changed. After you select one of these commands, the appropriate toggle option display is updated to reflect the change.

The P(rogram command is the command which is used to initiate the process of quickstarting a program. The operation of the P(rogram command is described in the following section.

The S(ystem command directs the QUICKSTART utility to build a description of the p-System operating system environment into a new system code file. This command is typically used only by sophisticated p-System users who are creating a new p-System operating system code file. The operation of the S(ystem command is basically the same as the operation of the P(rogram command described in the following section. The following section contains a supplemental description of the S(ystem command.

The Q(uit command is used to exit the QUICKSTART utility program.

The QUICKSTART utility menu is displayed after the completion of each P(rogram or S(ystem command.

The error messages which may be output by the QUICKSTART utility are listed and explained later. Generally, any error causes the processing associated with the current QUICKSTART command to be aborted and any output file discarded. QUICKSTART may occasionally generate warnings which appear in the form of a message on the console. These warning messages are also listed later in this chapter.

P(rogram Command

When the P(rogram command is entered, you are prompted:

```
Quickstart what program?
```

You should enter the name of the code file to be quickstarted (.CODE is appended to the name you enter if necessary). A plain <return> causes the current command to be canceled and the QUICKSTART menu to be displayed.

Once the input code file has been successfully opened, the action taken by QUICKSTART depends on the setting of the C(opy toggle option. If the C(opy toggle option is enabled, QUICKSTART prompts for the output file.

```
To what codefile?
```

An empty <return> cancels the current command and returns to the QUICKSTART menu. You may utilize the "$" character in the response to this prompt to denote the corresponding file name. For example, if the input file was 'MYDISK:BIGPROG.CODE' and your response to the above prompt is 'NEWDISK:$', QUICKSTART would generate the output file 'NEWDISK:BIGPROG.CODE'.

QUICKSTART automatically concatenates the suffix ".CODE" to the output file, unless you terminate the file name with a period. If you do terminate the file name with a period, however, a data file (rather than a code file) is created. You can create SYSTEM.PASCAL in this manner, but all other files must be created as code files (or they won't be executable).

Once the input file and the output file have been opened, QUICKSTART proceeds to create a copy of the original program code file. The code segments contained within the original program code file are copied one at a time and any old environment description for the program isn't copied.

When the M(essage toggle option is on, QUICKSTART displays a message at the start of the copying process which identifies the source and destination files involved in the copying. When the copying is completed, QUICKSTART displays the message "Copying complete." along with a report on the number of blocks which were copied.

Also, when the M(essages toggle option is on, the QUICKSTART utility displays messages which identify the names and library code file locations of the individual units and segments which are included in the description of the execution environment of the program. The following is an example of the messages that appear during a typical QUICKSTART P(rogram command:

```
Quickstart: P(rogram, S(ystem, C(opy, L(ibrary, M(essages, Q(uit [  ]
Toggle settings:  Copy ON,  Library copy OFF,  Messages ON
Quickstart what program? MYDISK:SUPERPROG.CODE
To what codefile? NEWDISK:$
Copying MYDISK:SUPERPROG.CODE to NEWDISK:SUPERPROG.CODE
Copying complete.  (278 blocks copied)
Using KERNEL from *SYSTEM.PASCAL
Including PROGINIT as segment of SUPERPRO from NEWDISK:SUPERPROG.CODE
Using SUPERPRO from NEWDISK:SUPERPROG.CODE
Using PASCALIO from *SYSTEM.PASCAL
Using HEAPOPS from *SYSTEM.PASCAL
Using PAGEMGR from ALTDISK:PAGEMGR.CODE
Installing new checksum into ALTDISK:EXPR.CODE
Installing new checksum into *SYSTEM.LIBRARY
Using LONGOPS from *SYSTEM.LIBRARY
Including FACTOR as segment of EXPR from ALTDISK:EXPR.CODE
Using EXPR from ALTDISK:EXPR.CODE
Quickstart construction complete.
Quickstart: P(rogram, S(ystem, C(opy, L(ibrary, M(essages, Q(uit [  ]
```

A message of the form "Using UNITNAME from FILE.NAME" reports the inclusion of the unit UNITNAME which is located in the code file FILE.NAME into the description of the execution environment for the program. A message of the form "Including SEGNAME as segment of UNITNAME from FILE.NAME" reports the inclusion of the segment SEGNAME as a part of the unit UNITNAME located in the library code file FILE.NAME.

A message of the form "Installing new checksum into FILE.NAME" informs you of the fact that QUICKSTART is attempting to install a checksum into library code file FILE.NAME.

When a library code file is updated with a new checksum and the L(ibrary copy toggle option is on, QUICKSTART asks you if a copy of the updated library code file is desired:

```
Copy updated file FILE.NAME?
```

This prompt is repeated until you respond with a 'Y' or 'N'. If you press 'Y', QUICKSTART prompts for the file to copy the updated library code file:

```
Copy to what codefile?
```

An empty <return> cancels the copying operation. The following is an example of a library code file copying operation during a P(rogram command:

```
Installing new checksum into RAMDISK:SYSTEM.LIBRARY
Copy updated file RAMDISK:SYSTEM.LIBRARY? Y
Copy to what codefile? MYDISK:$.
Copying RAMDISK:SYSTEM.LIBRARY to MYDISK:SYSTEM.LIBRARY
Copying complete.  (34 blocks copied)
```

S(ystem Command

The system command is used to quckstart the operating system (SYSTEM.PASCAL). This is intended to make the p-System boot more quickly.

NOTE: Although the S(ystem command is implemented within QUICKSTART, the operating system doesn't currently take advantage of it. This means the p-System will boot with the same speed whether or not the operating system is quickstarted. Quickstarting of the operating system will be supported in a future release of the p-System.

The S(ystem command directs QUICKSTART to install an environment description into a system code file presumed to contain the operating system. The operation of the S(ystem command is identical to the operation of the P(rogram command with the following exceptions:

- The generated environment description includes all of the units which reside in the system code file being processed, even if a subset of the units aren't referenced by the standard p-System units.

- The generated environment description doesn't contain references to the p-System code file in use at the time when the QUICKSTART utility is executed.

- An unresolved unit reference causes a warning message to appear on the console instead of resulting in a fatal error which terminates the processing associated with the command. This allows the p-System to contain references to units which provide the support for optional p-System components.

- The system code file must contain a unit with the name KERNEL, and that unit must have a subsidiary segment with the name USERPROG.

In the current p-System implementation, all of the units referenced within the operating system must reside in SYSTEM PASCAL. QUICKSTART doesn't enforce or check for this restriction however. In addition, QUICKSTART doesn't enforce or check for other implementation restrictions on the structure or type of units which can be placed in SYSTEM PASCAL.

Obsolete Environment Descriptions

Once an execution environment description is installed in a code file, it will be utilized to quickly construct the program's execution environment as long as the description doesn't become obsolete. An execution environment description becomes obsolete when one or more of the following alterations are made to the p-System environment in which the program is executed:

- SYSTEM.PASCAL is changed and the program contains a reference to an operating system unit which is no longer available.
- A referenced library code file is recompiled, reassembled, or altered using the p-System LIBRARY utility.
- A referenced library file can't be found after searching on the following volumes: the original volume where referenced, the prefix volume, the root volume.

Retention of the exact volume locations of referenced library code files result in optimal program invocation times. An individual library code file may be moved to a different physical location on the same volume without any resulting increase in program invocation time.

As mentioned previously, when an execution environment description becomes obsolete, it is still possible to execute the program. In such a situation, the p-System ignores the obsolete environment description and proceeds with the normal invocation of the program.

QUICKSTART Error Messages

The following is a list of the error messages which can be generated by the QUICKSTART utility program. Following each error message is a brief description of the error.

- Quickstart construction complete

  This is not an error message, but instead indicates successful completion of the QUICKSTART environment description generation process for a given program.

- Can't find FILE.NAME

  Indicates that the specified code file couldn't be found.

- Error reading library FILE.NAME

  An I/O error was detected by QUICKSTART when reading the specified library code file.

- Error inserting checksum into FILE.NAME

  An I/O error was detected by QUICKSTART when inserting a new checksum into the specified library code file.

- Error creating FILE.NAME

  An I/O error was detected by QUICKSTART when creating the indicated library code file copy.

- Error reading FILE.NAME

  An I/O error was detected by QUICKSTART when reading the indicated code file FILE.NAME.

- Error writing FILE.NAME

  An I/O error was detected by QUICKSTART when writing to the indicated code file.

- Library list file FILE.NAME isn't a text file

  The indicated file was specified as a library text file, but it isn't a text file.

- I/O error reading library list file FILE.NAME

  An I/O error was detected when reading the indicated text file which was specified as a library text file.

- Warning: Library FILE.NAME not found

  The indicated file was included on the library code file search list but couldn't be found. This is treated as a warning and not a fatal error since the missing library file is simply omitted from the list of library code files to search.

- Warning: UNIT_NAME unit not found

  The indicated unit is referenced by the system code file being processed by the QUICKSTART utility but can't be found. This is treated as a warning instead of a fatal error since the operating system is allowed to contain references to optional system units.

- Unit UNIT_NAME not found

  The indicated unit is required by the program being processed by QUICKSTART, but it can't be found within the program's code file or within one of the library code files.

- Duplicate unit UNIT_NAME

  This error indicates that there is more than one unit within the program's execution environment with the indicated name. This error can occur if there is more than one unit with the name within SYSTEM.PASCAL or when the name of the program is the same as the name of one of the units which reside in SYSTEM.PASCAL.

- Too many library code files referenced

  The required execution environment for the program contains references to more individual library code files than can be handled by the system. The current implementation allows an execution environment to contain references to at most 50 distinct library code files. This limitation can be worked around by using the LIBRARY utility to package several units into a single library code file. With the exception of SYSTEM.PASCAL, there is no limit on the number of units which can be packaged into a library code file.

- Too many system units referenced

  The required execution environment for the program contains references to more system units than can be handled by the system. A "system" unit is defined to be any unit which resides in the system code file SYSTEM.PASCAL. The current implementation allows an execution environment to contain references to at most 50 distinct system units.

- No program in code file to execute

  The code file to be executed doesn't contain a segment which is classified as being a host program. A unit by itself isn't an executable program. (This error can also appear when the QUICKSTART utility S(ystem command is used and the system code file being processed doesn't contain a unit with the name "KERNEL".)

- System code file doesn't contain a USERPROG segment

  This error message appears when the QUICKSTART utility S(ystem command is used and the system code file being processed doesn't contain a segment with the name "USERPROG".

- Unit UNIT_NAME must be linked via L(ink command

  The indicated unit contains references to assembly language routines which must be linked into the program by SYSTEM.LINKER before the program can be invoked.

- Segment SEG_NAME is an obsolete code segment

  The indicated code segment must be recompiled or reassembled with a more recent compiler or assembler before it can be executed on the current system.

- Insufficient memory to build environment

  The amount of available memory isn't sufficient to allocate the structures required to construct the execution environment for the program being invoked. The best work around for this situation is to reduce the number of separate library code files on the library code file search list and to reduce the total number of segment dictionary blocks which are contained within those library code files.

- Environment descriptor buffer overflow

  Internal error in the logic of the QUICKSTART utility.

REAL CONVERT

The REAL CONVERT utility can make some programs run more quickly. It converts real constants in a code file from canonical (compiled) form to native machine format. It eliminates the need to convert real constants at segment load time, thus increasing the initial loading speed of the program segments, as well as the overall run-time speed of the program. This is especially important for programs that require frequent loading of segments containing real constants.

The real constant conversion utility is a filter that works on code files, replacing canonical reals with run-time reals in-place. Hence, when the source file isn't available, you should make a backup copy of the code file to be processed before executing the utility program. This avoids the possibility of destroying the code file while executing REAL CONVERT with an unsuccessful write to disk.

Because the conversion algorithm uses real arithmetic of the host processor, the utility must be executed on the processor on which the output file will run. In most cases, a code file produced by the utility won't run on another processor, reducing the portability of otherwise transportable code.

To use the utility, X(ecute REALCONV from the Command menu. It responds with the following prompt:

```
ENTER FILE NAME:
```

Respond by entering the name of the code file to be processed, followed by <return>. You don't have to append the suffix .CODE.

If REAL CONVERT can't find the file, it prints the message 'File not found' and asks you to enter the file name again. Once a correct file is entered, REAL CONVERT begins translating.

If REAL CONVERT can't complete the conversion successfully, it prints a message and stops. The messages can be:

```
not enough memory

error in reading...
    The dots stand for:
      segment dictionaries
      first block
      constant pool
      segment
      (as the case may be).

error in writing segment

too many dictionaries
```

'Not enough memory' means that the segment to be processed is larger than the available memory space.

If the message is 'error in reading...', X(ecute REALCONV again.

If the message is 'error in writing segment', then, before X(ecuting REALCONV again, you have to restore the code file. Restoring the code file depends on the availability of the source file. If the source file is available, compile it again and save the code file. If only the code file was originally available, make a copy of the backup code file. (Remember to backup the original code file.)

'Too many dictionaries' means that you have more than 80 segments in the file.

The probability of getting any of the three messages is extremely slight, but it can happen.

If REAL CONVERT executes successfully, a dot is written on the console for each segment converted; and, once the conversion is completed, the message 'Enter file name:' is displayed so you can process another file. When there are no more files to process, answer the prompt by pressing <return>. This exits REAL CONVERT and returns you to the Command menu.

LIBRARY

LIBRARY.CODE is a utility program that allows you to group separate compilations (units or programs) and separately assembled routines into a single file. A library is a concatenation of such compilations and routines. Libraries are a useful means of grouping the separate pieces needed by a program or group of programs. Manipulating a single library file takes less time than if the various pieces it contains were each within an individual file. Libraries generally contain routines relating to a certain area of application; they can be used for functional groupings much as units can. Thus, you might want to maintain a math library, a data file-management library, and so forth—each of these libraries containing routines general enough to be used by many programs over a long period of time.

Individual programs might also take advantage of the library construct. If a program uses several units suitable for compiling separately, but the units themselves are too small to warrant putting each into its own file, you would want to construct a single library containing all of those units.

Even if a file contains only a single unit or routine, it is treated as a library when the unit or routine is used by some external host.

Library is useful for putting units into SYSTEM.LIBRARY or other libraries and grouping assembly routines together.

This section uses the term compilation unit. A program or unit and all the segments declared inside it are called a compilation unit. The segment for the program or unit is called the host segment of the compilation unit. Segment routines declared inside the host are called subsidiary segments. Units used by the host aren't segments belonging to that compilation unit. Units used by the compilation unit generate information in the host segment called segment references. The segment references contain the names of all segments referenced by a compilation unit, and the operating system uses this information to set up a run-time environment.

Some routines called from hosts exist in units in the operating system and, therefore, appear in segment references, even though there is no explicit USES declaration. For example, WRITELN resides in the operating system UNIT PASCALIO, so the name PASCALIO appears in the segment references of any host that calls WRITELN.

Using Library

When Library is executed, it displays a prompt asking for an output file name. The file name must end in .CODE. Library removes an old file with the same name as the new library.

Library then displays a prompt asking for the input file name. .CODE is automatically appended.

Library Example

You specify SCREENOPS.CODE as an input file. Library displays the following listing.

```
Library: N(ew, 0-9(slot-to-slot, E(very, S(elect, C(omp-unit, F(ill,?

Input file? SCREENOPS<return>
  0 u SCREENOP   582
  1 s SEGSCINI   508
  2 s SEGSCPRO   229
  3 s SEGSCCHE   126

Output file? NEW.CODE<return>
```

The preceding display shows that the file SCREENOPS consists of one unit and three segment routines. There are four possible types of code that can occupy the slots in a library: units, programs, segment routines, and assembled routines. Library displays the type, along with the name and length (in words) of each module.

Library's menu shows the various commands available.

- The N(ew command displays a prompt asking for a new input file.

- The A(bort command stops Library without saving the output file.

- The Q(uit command stops Library and saves the output file. Then Library displays the prompt, 'Notice?', at the top of the screen. Enter copyright notice and press <return>. It is placed in the output file's segment dictionary. Pressing <return> without entering a copyright notice exits Library without writing a copyright notice.

- The T(og command toggles a switch that determines whether or not INTERFACE parts of units are copied to the output file.

- The R(efs command lists the names of each entry in the segment reference lists of all segments currently in the output file. The list of names also includes the names of all compilation units currently in the output file, even though their names may not occur in any of the segment references.

The remaining five commands allow code segments to be transferred from the input file to the output file.

- A given slot can be transferred to the output file by entering a digit (0 through 9). Library then displays a prompt: 'Copy from slot # ?' along with the digit just entered. If that is the name of the slot, press <space>. If that is the first digit of a two-digit slot number, enter the second digit and press <space>. Library confirms the entry before actually copying code. Press <backspace> to correct errors. If you press <return> without entering a number, the copy doesn't happen and Library redisplays its menu.

  If the destination slot in the output file is already filled, the system displays a warning and no copy takes place. If an identical code segment is already present anywhere in the output file, the new code segment is copied anyway.

- The E(very command copies all of the codes in the input file to the output file. If, for any code segment, the corresponding slot in the output file is alread filled, then Library searches for the next available slot and places the code there. If, for any code segment, an identical code segment already exists in the output file, that segment isn't copied over.

- The S(elect command causes Library to display a prompt asking which code segments to transfer. For each code segment not already in the output file, Library displays the prompt: 'Copy from slot #_?'. Pressing 'Y' or 'N' causes the segment to be copied or passed by; pressing 'E' causes the remainder of the code segments to be transferred (as in E(very); pressing <space> or <return> aborts the S(elect. If the corresponding slot in the output file is filled, Library searches for the next available slot and places the code there.

- C(omp-unit causes Library to display the prompt: 'Copy what compilation unit?'. The compilation unit named is transferred along with any segment procedures that it references. Procedures already present in the output file aren't copied.

- F(ill does the equivalent of a C(omp-unit command for all the compilation units referenced by the segment references in the output file.

- I(nput displays the next page of the segment dictionary in the input file. (If there are more than 16 code segments in the file, two or more segment dictionary pages are required.)

- O(utput displays the next page of the segment dictionary in the output file.

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 creates a data file containing 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're sure that the new one is correct.

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:

1. Miscellaneous data about the system.

2. General information about the terminal.

3. Specific information about the terminal control keys.

Running SETUP

Run SETUP like any other program with the X(ecute command. 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 option, just press 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, don't underestimate the importance of using keys you can easily remember and of making dangerous keys, like BREAK, ESCAPE, and RUBOUT, hard to hit.

Once you have run SETUP, always backup SYSTEM.MISCINFO under another name. (OLD.MISCINFO is one suggestion.) You also might want to name your backups according to different terminals; for example, ADDS.MISCINFO, IQ120.MISCINFO, TELUD.MISCINFO, and so on. 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 won't 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).

The only thing SETUP won't arrange for you, as far as terminal handling goes, is to tell the system how to do random cursor positioning for your terminal. This is a feature that the Screen-Oriented Editor requires. To learn how to support this capability, see the section on the SCREENOPS unit in the Program Development Reference Manual. Also, see the section on GOTOXY in the Adaptable System Installation Manual.

Miscellaneous Notes for SETUP

In general, if SETUP prompts for a feature that your terminal doesn't have, set the item to NUL (zero).

Set your terminal to run in full duplex, with no auto-echo.

Don't use terminal functions that do a "delete and close up" on lines or characters—not all terminals have these functions, so they are supplied through the Screen-Oriented Editor's software.

You can use SETUP to specify two- or three-character control (escape) sequences from the terminal keyboard. For information concerning three-character escape sequences, see the Adapatable System Installation Manual under ANSI terminals.

If you use the ANSI SCREENOPS unit, instead of the standard SCREENOPS, the p-System ignores all of SETUP's screen parameters. They include:

BACKSPACE
ERASE LINE
ERASE SCREEN
ERASE TO END OF LINE
ERASE TO END OF SCREEN
LEAD IN TO SCREEN
MOVE CURSOR HOME
MOVE CURSOR RIGHT
MOVE CURSOR UP

In previous versions of the p-System, there were only 6 storage devices (4, 5, 9 through 12). The number of storage devices is now configurable with SETUP. After the highest-numbered storage device, subsidiary volumes are allocated device numbers. The number of subsidiary volumes is also configurable. Above the highest-numbered device set aside for subsidiary volumes, user-defined serial devices may be defined. The maximum number of user-defined serial devices is 16. The highest unit number allowed for any of these devices is 127. The following fields allocate these unit numbers:

FIRST SUBSIDIARY VOL NUMBER
MAX NUMBER OF SUBSIDIARY VOLS
MAX NUMBER OF USER SERIAL VOLS

These fields are described below.

The memory update feature of SETUP doesn't update any of the following fields:

HAS SPOOLING
HAS EXTENDED MEMORY
CODE POOL SIZE
CODE POOL BASE[FIRST WORD]
CODE POOL BASE[SECOND WORD]
SEGMENT ALIGNMENT
FIRST SUBSIDIARY VOL NUMBER
MAX NUMBER OF SUBSIDIARY VOLS
MAX NUMBER OF USER SERIAL VOLS

In order to update these fields, create a new SYSTEM.MISCINFO on the boot disk and reboot.

SYSTEM.MISCINFO — Data Items

The information in this section is very specific; you may skip it on first reading. However, if you have a question about a certain data item, look in this section. Default values are shown and, sometimes, are our recommendations. When no suggested values are given, you should consult your own terminal's documentation. The items are ordered according to SETUP's menu.

If you are using a hard copy terminal or a storage screen, rather than a CRT, you can ignore all the data items that are only used by the Screen-Oriented Editor, leaving them set to their defaults. In particular, if you are in doubt about a particular item, it is safest to leave it set to NUL. Always leave items set to NUL that concern features that your terminal doesn't have (ERASE LINE, for instance); the software takes care of these situations.

Please note that SETUP frequently distinguishes between a character that is a key on the keyboard and a character that is sent to the screen from the system; on some terminals, two different characters may perform the same function. On other terminals, the key pressed and the character sent for a given function may be the same.

There are a few characters you can't change with SETUP. These are CARRIAGE 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 screen, the backspace character should move the cursor one space to the left. Default: ASCII BS.

CODE POOL BASE[FIRST WORD]
CODE POOL BASE[SECOND WORD]

Use these two entries to determine where the code pool resides on machines that use extended memory.

On some extended memory systems, 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 least-significant four bits must always be 0 on 8086 systems. Depending upon your memory configuration, for 8086 systems you might set these values as follows:

```
FIRST WORD = 1
SECOND WORD = 0
```

This indicates the binary value of 1 followed by 16 zeros (the start of the second 64K area).

On 9900 systems, the FIRST WORD is the 990/10 memory BIAS. (This isn't a straight memory address; see your hardware manual for more information concerning 9900 BIAS.) It defines the start of the code pool area. The SECOND WORD isn't used. There is no error checking done on this value, anywhere, except by the 9900 hardware.

NOTE: The PoolBase field in the Pooldes record within the operating system will be set to the value indicated by these two fields. If the code pool is internal (that is, you aren't using extended memory), set both words to 0.

NOTE: Don't execute .RELPROC and .RELFUNC assembly language routines on TI 9900 systems when an external code pool is being used. Attempting to execute such a routine results in run-time error number 11 (instruction not implemented). Use .PROC and .FUNC, which forces code to be placed in the heap—instead of the external code pool.

CODE POOL SIZE

If the code pool is external, this entry indicates the number of WORDS, minus one, available for it to fill. The Poolsize field in Pooldes will be set to this value. This value may be as great as 32767 (a 64K area). It may also be smaller, if desired, but it should be at least 12287 (a 24K area). The base address of this area is given by the two code pool base words. This value is ignored if you aren't 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. Default: ASCII ETX.

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 activity and ignores whatever actions were taken. Default: ASCII ESC.

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 screen, this character erases all the characters on the line that the cursor is on.

ERASE SCREEN

When sent to the screen, this character erases the entire screen.

ERASE TO END OF LINE

When sent to the screen, 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 screen, this character erases all characters, starting at the current cursor position to the end of the screen.

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 p-System, only 6 storage devices were allowed: 4, 5, 9 through 12. Now the number of storage devices is configurable. The devices from 9 through "First subsidiary vol number" -1 are now standard storage devices. Subsidiary volumes start with the device number indicated by "First subsidiary vol number." The number of subsidiary volumes is determined by "Max number of subsidiary vols." The highest device number allowed for subsidiary volumes, standard storage devices, or user-defined serial volumes (described below) is 127. (The device numbers 128 and above are reserved for user-defined devices, as described under "The Extended SBIOS" in the Adaptable System Installation Manual.)

WARNING: "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 may be TRUE or FALSE. On PDP-11, LSI-11, 8080, Z-80, 6502, 8086, 8088, and HP86/87 processors this bit is FALSE. On the 68000, 9900, and 6809, it is TRUE.

HAS CLOCK

This value 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 doesn't have a clock, this must be FALSE. (If you are using the adaptable system, you must write your own clock-handler; until it is installed, this item must be FALSE.)

HAS EXTENDED MEMORY

When extended memory isn't 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 value may be TRUE or FALSE. If your terminal isn't a CRT, this should be FALSE.

HAS SLOW TERMINAL

This value 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 hasn't been LIBRARYed into SYSTEM.PASCAL, the p-System won't 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 hexadecimal (which throws away the eighth bit). For some keybords, which generate eight bit characters, use FF hexadecimal. Default: ASCII DEL.

KEY FOR BREAK

When this key is pressed while a program is running, the program terminates immediately with a run-time error. Recommendation: a key that is difficult to hit accidentally. Default: ASCII NUL.

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. Default: ASCII ACK.

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. Default: ASCII DC3.

KEY TO ALPHA LOCK

When sent to the screen, this character locks the keyboard in uppercase (alpha mode). It is usually a key on the keyboard as well. Default: ASCII DC2.

KEY TO DELETE CHARACTER

This deletes the character where the cursor is and moves the cursor one character to the left. Default: ASCII BS.

KEY TO DELETE LINE

This key deletes the line that the cursor is currently on. Default: ASCII DEL.

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:). Default: ASCII ETX.

KEY TO MOVE CURSOR DOWN
KEY TO MOVE CURSOR LEFT
KEY TO MOVE CURSOR RIGHT
KEY TO MOVE CURSOR UP

These keys are recognized by the Screen-Oriented Editor and are used when editing a document to move the cursor about the screen. 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

On some terminals, 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. The prefix is generated only as a lead in for characters where you have set PREFIXED[<item name>] to TRUE. An example of this is in MOVE CURSOR HOME.

MAX NUMBER OF SUBSIDIARY VOLS

This field indicates the maximum number of subsidiary volumes that may be mounted 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" -1. 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

This entry is the total number of user-defined serial volumes desired. The first device number assigned to a user-defined serial volume is "First subsidiary vol number" + "Max number of subsidiary vols."

For example, if "First subsidiary vol number" is 12 (#12:) and "Max number of subsidiary vols" is 4, then the first user-defined serial volume #16:. If this entry, "Max number of user serial vols", is 2, then the user-defined serial volumes are #16: and #17:.

If "Max number of subsidiary vols" is 0, then the first user-defined serial volume is equal to "First subsidiary vol number". In this case, "Max number of user serial vols" + "First subsidiary vol number" -1 yields the highest-numbered user-defined serial volume.

NOTE: The largest value allowed for "Max number of user serial vols" is 16. The highest-numbered user-defined serial volume must be less than or equal to 127.

NOTE: User-defined serial volumes differ from user-defined devices (described under "The Extended SBIOS" in the Adaptable System Installation Manual). User-defined serial volumes are part of the system devices. These devices are allocated device numbers 0 through 127. Device numbers 128 through 255 are allocated for true user-defined devices. User-defined devices can only be accessed using unit I/O, whereas the standard p-System file I/O capabilities can be used with system devices such as user-defined serial volumes.

MOVE CURSOR HOME

When sent to the terminal, this key moves the cursor to the upper left of the screen (position (0,0)). If your terminal doesn't have a character that does this, this data item must be set to CARRIAGE RETURN; then, you won't be able to use the Screen-Oriented Editor.

MOVE CURSOR RIGHT

When sent to the terminal, this moves the cursor nondestructively one space to the right. If your terminal doesn't have this function, you won't be able to use the Screen-Oriented Editor.

MOVE CURSOR UP

When sent to the terminal, this moves the cursor up one line. If your terminal doesn't have this function, you won't be able to use the Screen-Oriented Editor.

NONPRINTING CHARACTER

This character is displayed on the screen when a nonprinting character is entered 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 screen 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 defined as an echoable code.

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 (32..126) includes the values 32, 126, and all values between them. The default is:

13, 32..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 screen. If you are using a hard copy terminal, set this to 0.

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 (that is, 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.

The processor segment alignments are as follows:

	Non-extended Memory	Extended Memory
Z80	0	N/A
8080	0	N/A
8086	0	16
9900	0	0
6502	0	N/A
6809	0	N/A
68000	0	0
HP-87	0	0
PDP-11tm	0	64

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[NONPRINTING CHARACTER]
PRINTABLE CHARACTERS
SCREEN HEIGHT
SCREEN WIDTH
SEGMENT ALIGNMENT
STUDENT
VERTICAL MOVE DELAY

Sample SETUP Session

The following is a sample of part of a SETUP session. The data is being changed from the system defaults to the specifications for a Soroc terminal. All underlined text like this you enter, and all text enclosed in curly brackets {like this} is commentary. Angle brackets <these> are used to enclose the names of nonprinting characters {like <return>}. All else is SETUP's output to the terminal.

```
{To begin, you must eXecute SETUP}

XSETUP<return>
INITIALIZING.........................
.........................
SETUP: C(HANGE T(EACH H(ELP Q(UIT [D1]

{H(ELP tells you about the other activities, and T(EACH
 describes the use of SETUP.  Now is the most profitable
 time to use these activities.
 Suppose you have read H(ELP and T(EACH, and decide to
 change data items by going through the menu.  You must
 press 'C' for C(HANGE.}

C

{Note:  these single-character activities don't echo.}

CHANGE: S(INGLE) P(ROMPTED) R(ADIX)
    H(ELP) Q(UIT)

{H(ELP) describes the activities on this particular line,
 R(ADIX) allows you to change the base of the numbers
 you enter, and Q(UIT) returns you to the SETUP: menu.
 What you want to do now is go through the prompted menu.}

P

FIELD NAME = BACKSPACE
 OCTAL  DECIMAL  HEXADECIMAL  ASCII  CONTROL
   10       8        8         BS      ^H
WANT TO CHANGE THIS VALUE? (Y,N,!)
<return>
WANT TO CHANGE THIS VALUE? (Y,N,!)

{<return> or <space> will cause this menu to be repeated.
 '!' causes an escape to the CHANGE: menu.
 Since control-H (^H) is indeed the Soroc's backspace,
 you want to go on.}
```

```
N

FIELD NAME = EDITOR ACCEPT KEY
 OCTAL  DECIMAL  HEXADECIMAL  ASCII  CONTROL
    0       0       0         NUL     ^@
WANT TO CHANGE THIS VALUE? (Y,N,!)

Y

NEW VALUE: <home>

{When <home> or any other nonprinting key
 is pressed, '?' is displayed.}

OCTAL  DECIMAL  HEXADECIMAL  ASCII  CONTROL
    3       3       3         ETX     ^C
WANT TO CHANGE THIS VALUE? (Y,N,!)

N

FIELD NAME = EDITOR ESCAPE KEY
 OCTAL  DECIMAL  HEXADECIMAL  ASCII  CONTROL
    0       0       0         NUL     ^@
WANT TO CHANGE THIS VALUE (Y,N,!)

Y

NEW VALUE: <return>

{Any unexpected input here causes the
 relevant section of T(EACH to be output,
 followed by this:}
C(ONTINUE)
{All characters are ignored except 'C', and
 then the menu is repeated.}

C

NEW VALUE: <rubout> {Again, a '?' is echoed.}

OCTAL  DECIMAL  HEXADECIMAL  ASCII
  177     127     7F         DEL
WANT TO CHANGE THIS VALUE? (Y,N,!)

{(Note that there is no corresponding control key.)
  DEL is not the key you meant, so you must
  change it again.}

Y

NEW VALUE: <esc> {? is echoed.}
 OCTAL  DECIMAL  HEXADECIMAL  ASCII  CONTROL
   33      27      1B         ESC     ^[
WANT TO CHANGE THIS VALUE? (Y,N,!)

N
```

```
{This is what it should be.}

{The menu continues in this way for the rest of
 the data items.  Suppose you have gone ahead and
 answered all of the questions according to the
 Soroc specifications; after the last data item,
 you again get the menu:}

CHANGE: S(INGLE) P(ROMPTED) R(ADIX)
    H(ELP) Q(UIT)

{You realize that you left the prefix for
 ERASE LINE at FALSE, when it should be
 TRUE.  You want to change just this one
 data item.}

S {For S(INGLE)}

NAME OF FIELD: PREFIXED [ERASE]
DIDN'T FIND PREFIXED [ERASE]    {Oops}
NAME OF FIELD: PREFIXED [ERASE LINE]

FIELD NAME = PREFIXED [ERASE LINE]
 CURRENT VALUE IS FALSE
WANT TO CHANGE THIS VALUE? (Y,N,!)

Y

NEW VALUE: TRUE {T would also work.}
 CURRENT VALUE IS TRUE
WANT TO CHANGE THIS VALUE? (Y,N,!)

N

CHANGE: S(INGLE) P(ROMPTED) R(ADIX)
    H(ELP) Q(UIT)

Q

SETUP: C(HANGE T(EACH H(ELP Q(UIT [D2]

Q {You're through changing data now.}

QUIT: D(ISK) OR M(EMORY) UPDATE,
         R(ETURN) H(ELP) E(XIT)

{You want to do a disk update to create
 NEW.MISCINFO on your disk for future use.}

D

QUIT: D(ISK) OR M(EMORY) UPDATE,
         R(ETURN) H(ELP) E(XIT)

E

{And now you're done.  The Command menu
 will appear.}
```

Sample Terminal Setups

Here is a list of SYSTEM.MISCINFO data items followed by some sample values for four popular terminals. Some items in the SETUP menu haven't been included; these are data items that refer to your processor configuration, not your terminal.

These examples represent what we consider reasonable layouts for a few different keyboards, but we don't guarantee that they work for your particular hardware or match your individual taste.

Terminals:	LSI ADM-3A	HAZELTINE 1500/1510	SOROC IQ120	HEATH H19	ADDS Viewpoint 3A plus
Data Items:					
BACKSPACE	left-arrow	backspace	ctrl-H	ctrl-H	ctrl-H
EDITOR ACCEPT KEY	ctrl-C	ctrl-C	home	ctrl-C	ctrl-C
EDITOR ESCAPE KEY	esc	esc	esc	ctrl-[	esc
ERASE LINE	NUL	NUL	NUL	l	NUL
ERASE SCREEN	ctrl-Z	ctrl-\	'*'	E	'*'
ERASE TO END OF LINE	NUL	ctrl-O	T	K	T
ERASE TO END OF SCRN	NUL	ctrl-X	Y	J	Y
HAS LOWER CASE	TRUE	TRUE	TRUE	TRUE	TRUE
HAS RAND CURS ADDR	TRUE	TRUE	TRUE	TRUE	TRUE
HAS SLOW TERMINAL	FALSE	FALSE	FALSE	FALSE	FALSE
KEY FOR BREAK	ctrl-B *	break **	break	break	ctrl-B
KEY FOR FLUSH	ctrl-F	ctrl-F	ctrl-F	ctrl-F	ctrl-F
KEY FOR STOP	ctrl-S	ctrl-S	ctrl-S	ctrl-S	ctrl-S
KEY TO ALPHA LOCK	ctrl-R	NUL	ctrl-R	ctrl-R	ctrl-R
KEY TO DELETE CHAR	ctrl-H	backspace	l-arrow	ctrl-H	ctrl-H
KEY TO DELETE LINE	rubout	shift-DEL	rubout	DEL	DEL
KEY TO END FILE	ctrl-C	ctrl-C	ctrl-C	ctrl-C	ctrl-C
KEY TO MV CURS DOWN	ctrl-J	ctrl-K	d-arrow	B	ctrl-J
KEY TO MV CURS LEFT	ctrl-H	backspace	l-arrow	D	ctrl-H
KEY TO MV CURS RGHT	ctrl-L	ctrl-P	r-arrow	C	ctrl-L
KEY TO MV CURS UP	ctrl-K	ctrl-L	u-arrow	A	ctrl-K
LEAD IN FROM KEYBD	NUL	NUL	NUL	esc	NUl
LEAD IN TO SCREEN	NUL	~	esc	esc	esc
MOVE CURSOR HOME	ctrl-^	ctrl-R	ctrl-^	H	ctrl-^
MOVE CURSOR RIGHT	ctrl-L	ctrl-P	r-arrow	C	ctrl-L
MOVE CURSOR UP	ctrl-K	ctrl-L	u-arrow	A	ctrl-K
NONPRINTING CHAR	'?'	'?'	'?'	'?'	'?'
PREF [DELETE CHAR]	FALSE	FALSE	FALSE	FALSE	FALSE
PREF [ED ACCEPT KEY]	FALSE	FALSE	FALSE	FALSE	FALSE
PREF [ED ESCAPE KEY]	FALSE	FALSE	FALSE	TRUE	FALSE
PREF [ERASE LINE]	FALSE	FALSE	FALSE	TRUE	FALSE
PREF [ERASE SCREEN]	FALSE	TRUE	TRUE	TRUE	FALSE
PREF [ERASE TO EOLN]	FALSE	TRUE	TRUE	TRUE	TRUE
PREF [ERSE TO EOSCN]	FALSE	TRUE	TRUE	TRUE	TRUE
PREF [KEY DEL CHAR]	FALSE	FALSE	FALSE	FALSE	FALSE
PREF [KEY DEL LINE]	FALSE	FALSE	FALSE	FALSE	FALSE
PREF [KEY MV CRS DN]	FALSE	FALSE	FALSE	TRUE	FALSE
PREF [KEY MV CRS LT]	FALSE	FALSE	FALSE	TRUE	FALSE
PREF [KEY MV CRS RT]	FALSE	FALSE	FALSE	TRUE	FALSE
PREF [KEY MV CRS UP]	FALSE	FALSE	FALSE	TRUE	FALSE
PREF [MOVE CRS HOME]	FALSE	TRUE	FALSE	TRUE	FALSE
PREF [MOVE CURS RT]	FALSE	FALSE	FALSE	TRUE	FALSE
PREF [MOVE CURS UP]	FALSE	FALSE	FALSE	TRUE	FALSE
PREF [NONPRINT CHAR]	FALSE	FALSE	FALSE	FALSE	FALSE
SCREEN HEIGHT	24	24	24	24	24
SCREEN WIDTH	80	80	80	80	80
STUDENT	FALSE	FALSE	FALSE	FALSE	FALSE
VERTICAL MOVE DELAY	5	5	10	10	0

* The BREAK key can also be used, but it's perilously close to RETURN.

** Break is also control-@ on Hazeltines.

Terminals:	DEC VT-52	IBM PC	DATA-MEDIA
Data Items:			
BACKSPACE	backspace	ctrl-H	backspace
EDITOR ACCEPT KEY	ctrl-C	ctrl-C	ctrl-C
EDITOR ESCAPE KEY	esc	esc	esc
ERASE LINE	ctrl-@	L	ctrl-@
ERASE SCREEN	ctrl-@	E	ctrl-L
ERASE TO END OF LINE	K	K	ctrl-]
ERASE TO END OF SCRN	J	J	ctrl-K
HAS LOWER CASE	TRUE	TRUE	TRUE
HAS RAND CURS ADDR	TRUE	TRUE	TRUE
HAS SLOW TERMINAL	FALSE	FALSE	FALSE
KEY FOR BREAK	ctrl-@	ctrl-_	ctrl-@
KEY FOR FLUSH	ctrl-F	ctrl-F	ctrl-F
KEY FOR STOP	ctrl-S	ctrl-S	ctrl-S
KEY TO ALPHA LOCK	ctrl-R	ctrl-R	ctrl-R
KEY TO DELETE CHAR	ctrl-H	ctrl-H	backspace
KEY TO DELETE LINE	del	del	del
KEY TO END FILE	ctrl-C	ctrl-C	ctrl-C
KEY TO MV CURS DOWN	B	B	d-arrow
KEY TO MV CURS LEFT	D	D	l-arrow
KEY TO MV CURS RGHT	C	C	r-arrow
KEY TO MV CURS UP	A	A	u-arrow
LEAD IN FROM KEYBD	esc	ctrl-Q	ctrl-@
LEAD IN TO SCREEN	esc	esc	ctrl-@
MOVE CURSOR HOME	H	H	ctrl-Y
MOVE CURSOR RIGHT	C	C	ctrl-\
MOVE CURSOR UP	A	A	ctrl-_
NONPRINTING CHAR	'?'	'?'	'?'
PREF [DELETE CHAR]	FALSE	FALSE	FALSE
PREF [ED ACCEPT KEY]	FALSE	FALSE	FALSE
PREF [ED ESCAPE KEY]	TRUE	FALSE	FALSE
PREF [ERASE LINE]	FALSE	TRUE	FALSE
PREF [ERASE SCREEN]	FALSE	TRUE	FALSE
PREF [ERASE TO EOLN]	TRUE	TRUE	FALSE
PREF [ERSE TO EOSCN]	TRUE	TRUE	FALSE
PREF [KEY DEL CHAR]	FALSE	FALSE	FALSE
PREF [KEY DEL LINE]	FALSE	FALSE	FALSE
PREF [KEY MV CRS DN]	TRUE	TRUE	FALSE
PREF [KEY MV CRS LT]	TRUE	TRUE	FALSE
PREF [KEY MV CRS RT]	TRUE	TRUE	FALSE
PREF [KEY MV CRS UP]	TRUE	TRUE	FALSE
PREF [MOVE CRS HOME]	TRUE	TRUE	FALSE
PREF [MOVE CURS RT]	TRUE	TRUE	FALSE
PREF [MOVE CURS UP]	TRUE	TRUE	FALSE
PREF [NONPRINT CHAR]	FALSE	FALSE	FALSE
SCREEN HEIGHT	24	25	24
SCREEN WIDTH	80	80	80
STUDENT	FALSE	FALSE	FALSE
VERTICAL MOVE DELAY	0	0	0

DISKSIZE

The DISKSIZE utility enables you to alter the storage capacity of a disk without having to change the files on it. For example, you could change a disk's size from 640 blocks to 320 blocks.

When you use DISKSIZE to make a disk smaller, you should be sure that there is enough unused space after the last file to absorb the decrease in storage capacity. If there isn't, the resulting directory will be internally inconsistant since disk space is being used which isn't supposed to be available. (Files aren't removed by DISKSIZE.) If you attempt to use DISKSIZE to make a disk larger than its maximum storage capacity, DISKSIZE will inform you that this can't be done.

When you X(ecute DISKSIZE, these prompts appear:

```
Change directory size on what unit (4,5,9..22) ?
Current size is xxx blocks
What is new directory size in 512 byte blocks ?
```

In response to the first prompt, you should enter the device number of the disk to be altered. (Don't include the # sign or the colon in the device number.)

The second line indicates the size of the disk according to the current directory. The "xxx" is actually a number such as 320.

The last prompt asks you to enter the new size for the disk. You should enter the desired number followed by <return>. If the number you enter is larger than the maximum capacity (or smaller than the minimum capacity) of the disk, you are prompted:

```
No such block
What is new directory size in 512 byte blocks ?
```

This means that you entered an invalid number and should try again.

COPYDUPDIR

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 doesn't currently contain a duplicate directory, COPYDUPDIR displays a prompt stating that. 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 'Y' to execute the copy; any other character aborts the program.

MARKDUPDIR

MARKDUPDIR creates a duplicate directory on a disk that doesn't currently contain one.

Be sure that blocks 6 through 9 are free for use. If they aren't, 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 doesn't have a duplcate directory. However, if the first file or unused area starts at block 10, then the disk already has a duplicate directory.

MARKDUPIR Example

```
SYSTEM.PASCAL        106   1-Jan-83     6    Codefile
.
.
.
OR

<unused>               4                 6
SYSTEM.PASCAL        106   1-Jan-83    10    Codefile
.
.
.
```

Both of the preceding cases indicate disks that have no duplicate directory. The following listing is a directory of a properly marked disk:

```
SYSTEM.PASCAL        106   1-Jan-83    10    Codefile
.
.
.
```

To create a duplicate directory, X(ecute '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 aren't, the system displays a prompt asking if you are sure they are free. Press 'Y' 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'll lose file information.

RECOVER

The RECOVER utility attempts to recreate the directory of a disk whose directory has accidentally been destroyed.

When you X(ecute 'RECOVER', it prompts you for the drive number of the disk you wish to recover:

```
Recover [version]
USER'S DISK IN DRIVE # (0 exits):
```

You should enter the number, such as '5', without the pound sign or colon, followed by <return>. Next, you are prompted for the new name to be given to the recovered volume:

```
USER'S VOLUME ID:
```

You should enter a correct volume name. Finally, RECOVER prompts:

```
How many blocks on disk?
```

Here you should indicate the total number of blocks on the volume being recovered.

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 'N', RECOVER displays the following prompt:

```
GO AHEAD AND UPDATE DIRECTORY (Y/N)?
```

If you press 'N', RECOVER finishes executing without doing anything.

If you press 'Y', RECOVER saves the reconstructed directory and display the following prompt:

```
WRITE OK
```

Then RECOVER terminates.

If you press 'Y' 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 can't determine the original name of a 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 can't detect data files since their format isn't system-defined. To recover data files, you must use the PATCH utility, described in the 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 use the linker to relink recovered code files (if linking was originally necessary).

When RECOVER has finished its pass over the entire disk, it displays the following prompt:

```
GO AHEAD AND UPDATE DIRECTORY (Y/N)?
```

Appendices

APPENDICES

APPENDIX A
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, Break Point
15	Bad Block
16	Break Point
17	Incompatible 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 rebootstrap. Some FATAL errors leave the system in an irreparable state, in which case the user must rebootstrap.

APPENDIX B
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
19	Bad text file size

APPENDIX C
DEVICE NUMBERS

Device Number	Volume Name
1	CONSOLE:
2	SYSTERM:
4	<System disk '*'>
5	<other disk>
6	PRINTER:
7	REMIN:
8	REMOUT:
9...127	<additional disks, subsidiary volumes, or user-defined serial devices>
128...255	<user-defined devices>

APPENDIX D
ASCII TABLE

0	000	00	NUL	32	040	20	SP	64	100	40	@	96	140	60	`
1	001	01	SOH	33	041	21	!	65	101	41	A	97	141	61	a
2	002	02	STX	34	042	22	"	66	102	42	B	98	142	62	b
3	003	03	ETX	35	043	23	#	67	103	43	C	99	143	63	c
4	004	04	EOT	36	044	24	$	68	104	44	D	100	144	64	d
5	005	05	ENQ	37	045	25	%	69	105	45	E	101	145	65	e
6	006	06	ACK	38	046	26	&	70	106	46	F	102	146	66	f
7	007	07	BEL	39	047	27	'	71	107	47	G	103	147	67	g
8	010	08	BS	40	050	28	(	72	110	48	H	104	150	68	h
9	011	09	HT	41	051	29	)	73	111	49	I	105	151	69	i
10	012	0A	LF	42	052	2A	*	74	112	4A	J	106	152	6A	j
11	013	0B	VT	43	053	2B	+	75	113	4B	K	107	153	6B	k
12	014	0C	FF	44	054	2C	,	76	114	4C	L	108	154	6C	l
13	015	0D	CR	45	055	2D	-	77	115	4D	M	109	155	6D	m
14	016	0E	SO	46	056	2E	.	78	116	4E	N	110	156	6E	n
15	017	0F	SI	47	057	2F	/	79	117	4F	O	111	157	6F	o
16	020	10	DLE	48	060	30	0	80	120	50	P	112	160	70	p
17	021	11	DC1	49	061	31	1	81	121	51	Q	113	161	71	q
18	022	12	DC2	50	062	32	2	82	122	52	R	114	162	72	r
19	023	13	DC3	51	063	33	3	83	123	53	S	115	163	73	s
20	024	14	DC4	52	064	34	4	84	124	54	T	116	164	74	t
21	025	15	NAK	53	065	35	5	85	125	55	U	117	165	75	u
22	026	16	SYN	54	066	36	6	86	126	56	V	118	166	76	v
23	027	17	ETB	55	067	37	7	87	127	57	W	119	167	77	w
24	030	18	CAN	56	070	38	8	88	130	58	X	120	170	78	x
25	031	19	EM	57	071	39	9	89	131	59	Y	121	171	79	y
26	032	1A	SUB	58	072	3A	:	90	132	5A	Z	122	172	7A	z
27	033	1B	ESC	59	073	3B	;	91	133	5B	[	123	173	7B	{
28	034	1C	FS	60	074	3C	<	92	134	5C	\	124	174	7C	|
29	035	1D	GS	61	075	3D	=	93	135	5D	]	125	175	7D	}
30	036	1E	RS	62	076	3E	>	94	136	5E	^	126	176	7E	~
31	037	1F	US	63	077	3F	?	95	137	5F	_	127	177	7F	DEL

APPENDIX E
CONFIGURATION NOTES

This appendix briefly covers several topics related to p-System configuration and possible problems that you might encounter.

FLOATING POINT PACKAGES

The p-System may be configured to run with two-word real numbers (32 bit precision), four-word real numbers (64 bit precision), or no floating point arithmetic at all. Programs which use two-word precision can perform floating point operations with 6 to 7 digits of accuracy and a base 10 exponent with an absolute value as large as 38 (approximately). Programs using four-word precision can have up to 15 or 16 digits of precision and a base 10 exponent with an absolute value as large as 308 (approximately). These values vary somewhat among processors.

The memory available to p-System programs decreases as you go from no reals, to two-word reals, to four-word reals. This results in tradeoffs between floating point precision and memory space availability that you should take into consideration. Execution speed may also be a factor since code may have to be swapped to and from disk more often when there is less main memory space available.

Application programs which use floating point arithmetic require a p-System configured with the same real size that they use. If you attempt to run a program which uses a different real size from the p-System you are using, you will receive a real size mismatch error (execution error 17). If you attempt to run a program which uses reals on a p-System configured for no reals, you will receive an unimplemented instruction error (execution error 11). (Programs that don't use real numbers will run regardless of the floating point precision of the p-System.)

When a Pascal program is compiled, the compiler creates a code file which has the real size of the PME being used. If you want to create a code file with a specific real size (which doesn't necessarily correspond to the underlying PME at compilation time), you can use the $R compiler option (described in the Program Development Reference Manual). For FORTRAN and BASIC, there is a two-word and four-word version of the compiler. You should choose the compiler which produces the desired real number size.

The rest of this section outlines how to create a system with the real size that you want. This information may not apply to you, however, since some p-System suppliers provide separate boot disks which are already configured for the different real sizes.

The operating system (SYSTEM.PASCAL) and the p-machine emulator (usually called SYSTEM.INTERP) both must be configured for a particular real size. They should be configured consistently with each other.

In order to create a new SYSTEM.PASCAL with real numbers, you must use the LIBRARY utility (described in Chapter 5). With this utility, you should place the appropriate REALOPS unit into the new SYSTEM.PASCAL. Whenever you use the LIBRARY utility to create a SYSTEM.PASCAL, you must be sure that the segments KERNEL and USERPROG remain in slots 0 and 15, respectively. It is a good idea to first move REALOPS over to a slot greater than 15. After that, move all of SYSTEM.PASCAL over to the new file.

Here is a step-by-step process that you may follow if you are unfamiliar with the Library utility:

1. Make sure that you have a disk with enough free space (approximately 120 contiguous blocks) to contain the new SYSTEM.PASCAL. You should be sure that a SYSTEM.PASCAL doesn't already exist on that disk. For this discussion, that disk will be called "NEW_PAS:".

2. Locate the appropriate REALOPS code file (for two-word or four-word reals). Place this code file on NEW_PAS: using the filer's T(ransfer facility. For this discussion, that file will be referred to as "REALOPS.CODE" even though it actually has a slightly different name (depending upon your processor and real size).

3. Locate the disk which contains the LIBRARY utility and place it in drive #5.

4. From the Command menu, X(ecute #5:LIBRARY.

5. After LIBRARY's first prompt appears on the screen, remove the disk from #5 and place NEW_PAS: in #5. Be sure that the system disk is in #4.

6. Respond to LIBRARY's prompts like this:

```
Output file? NEW_PAS:SYSTEM.PASCAL <ret>
Input file? NEW_PAS:REALOPS.CODE <ret>
TYPE: 'T'
TYPE: '0 <space> 21 <space>
TYPE: 'N'
Input file? *SYSTEM.PASCAL <ret>
TYPE: 'E'
TYPE: 'Q'
TYPE: <ret>
```

7. NEW_PAS:SYSTEM.PASCAL is now configured for the appropriate real size. Later, you should T(ransfer this file to a bootable disk.

After you have created the new SYSTEM.PASCAL, the next step is to locate the PME which has the real size that you want. You are provided with two-word and four-word PMEs in addition to a PME which doesn't support real numbers. You should simply locate the appropriate PME code file for now. Later you should use T(ransfer to move that code file to the boot disk giving it the name SYSTEM.INTERP. (On some systems, the PME is given a different name, such as SYSTEM.PDP-11, SYSTEM.IBM, and so forth).

In order to create a bootable disk which contains the new SYSTEM.PASCAL and SYSTEM.INTERP, you need to follow a process which is machine-specific. Here is a general outline of the necessary steps:

1. Format a new diskette.

   This involves executing a disk formatter program which should be described in your machine-specific documentation. Not all computers require that a diskette be formatted, however.

2. Initialize the p-System directory.

   You can do this using the filer's Z(ero command (described in Chapter 2). Some disk formatter programs do this automatically, however.

3. Use T(ransfer to move the necessary system files onto the new boot disk.

   These files include SYSTEM.PASCAL, SYSTEM.INTERP, and SYSTEM.MISCINFO. (Some systems require additional files such as SYSTEM.BIOS, SYSTEM.SBIOS, or SYSTEM.BOOT. If this is necessary on your computer, your machine-specific documentation should explain it.) Usually, SYSTEM.LIBRARY is kept on the boot disk as well.

4. Place the bootstrap code on the new diskette.

   The bootstrap code (which is only required on bootable disks) resides in an area of the disk which doesn't appear in the p-System directory. In order to place this code on a disk, you may need to use a special utility program such as BOOTER. Some disk formatter programs automatically place the bootstrap on the newly formatted diskette. A volume-to-volume T(ransfer will copy the bootstrap code on many computers. (If this is the case with your computer, you can, if you wish, just T(ransfer two blocks from a bootable disk to a new disk. This will copy the bootstrap code without disturbing the directory or any files that may already reside on the new disk.) The process of placing a bootstrap on a new disk should be described in your machine-specific documentation.

THE DEBUGGER

The debugger is described in the Program Development Reference Manual. It can be used as an aid in debugging programs that you develop. In order to use the debugger, you may have to use the LIBRARY utility to place DEBUGGER.CODE into SYSTEM.PASCAL. (See the "Floating Point Packages" section, above, about using LIBRARY to create a new SYSTEM.PASCAL.)

In order to use the symbolic debugging facility, you may also have to place the symbolic debugging unit (usually found in PDBG.SEED.CODE) into SYSTEM.PASCAL using LIBRARY.

The reason that the debugger isn't necessarily placed in SYSTEM.PASCAL is that it requires extra disk space and not all p-System users need it.

You should be aware that if you select D(ebug and there is no debugger in your system, the p-System will halt. It is necessary to reboot under these circumstances.

EXTENDED MEMORY

Extended memory is a feature that allows the p-System to run in environments of up to 128K bytes (or more) of memory. This is accomplished by dividing the p-System run-time environment into two parts, each of which may occupy as much as 64K bytes of memory. (On many computers, a RAM disk can be used if you have more than 128K.)

The code pool is an area of memory where most code segments are executed by the p-System. This code includes the operating system, filer, editor, and so on, as well as your programs. In nonextended memory systems, the code pool shares the same space with the rest of the p-System (for example, the interpreter, RSP, BIOS, SBIOS, and the p-System stack and heap). The code pool resides between the stack and heap on nonextended memory systems.

On extended memory systems, the code pool is placed in a separate area of memory altogether. Thus, the code pool may occupy an entire 64K portion of RAM, and the rest of the p-System may occupy another entire 64K area.

A major advantage of the extended memory feature is the additional memory space available for executable code to use. This means that larger programs can be compiled and executed.

Also, the code segments on extended memory systems may not need to be moved or swapped as often as those on nonextended memory systems thereby producing significant performance improvements.

Because there is more space for the p-System stack and heap to grow, the chances of a stack overflow are reduced.

The SYSTEM.MISCINFO item "HAS EXTENDED MEMORY" must be set to true if you are using extended memory, and false, otherwise. If HAS EXTENDED MEMORY isn't set correctly, the p-System won't boot.

BOOTING PROBLEMS

If you are having problems bootstrapping the p-System, there are several simple mistakes that you may have made. This section briefly covers them. An appendix to the Adaptable System Installation Manual covers some more detailed problems that you might encounter when attempting to bootstrap an adaptable system.

- You may have forgotten to place a bootstrap on the disk. The bootstrap code doesn't appear in the directory because it resides in an area outside of the main p-System volume (usually in the first two blocks of the first p-System track on the diskette). A bootstrap is placed on the diskette in a machine-specific manner. On some machines the BOOTER utility is used. On PDP-11 or LSI-11 machines, the ABOOTER utility is used. (BOOTER and ABOOTER are described in the Adaptable System Installation Manual.) Other implementations use a special utility program to copy a bootstrap onto a new diskette (often in conjunction with disk formatting and directory initialization).

- You may not have all the necessary system files on the diskette. SYSTEM.PASCAL, SYSTEM.INTERP, and SYSTEM.MISCINFO all must be on the system disk if it is to bootstrap successfully. (Actually, SYSTEM.INTERP may have another name on your particular system.) Some systems require other files on the system disk such as SYSTEM.BIOS, SYSTEM.SBIOS, or SYSTEM.BOOT.

- Any of the following fields in SYSTEM.MISCINFO may have been set incorrectly:

  CODE POOL BASE
  CODE POOL SIZE
  HAS EXTENDED MEMORY
  HAS SPOOLING
  SEGMENT ALIGNMENT

If any of these are incorrectly set, the system may not boot. You should be sure that you keep a backup copy of any system disk which does boot successfully (since you need to boot the p-System in order to alter SYSTEM.MISCINFO with the SETUP utility).

SCREEN DISPLAY PROBLEMS

If your screen doesn't display information correctly, there are two likely problems:

- SYSTEM.MISCINFO is incorrectly configured for your console. In the section on SETUP (in Chapter 5) several sample terminal setups are given. Any of the SYSTEM.MISCINFO items shown there may adversely affect the screen display if they are set incorrectly for your hardware. Note that the four SYSTEM.MISCINFO items which begin with "ERASE" may be especially troublesome if set incorrectly. If these "ERASE" items are set to NUL (ASCII 0), then the p-System will function correctly (but slower than if they are set to the correct values for your hardware). However, if both ERASE LINE and ERASE TO END OF LINE are set to NUL, the display won't always be correct.

- You have an incorrect GOTOXY unit within SYSTEM.PASCAL. GOTOXY moves the cursor to a given "X" and "Y" coordinate on the screen. Different terminals perform this in different ways, so GOTOXY is terminal-dependent. See the Adaptable System Installation Manual for more information about GOTOXY.

APPENDIX F
USUS MEMBERSHIP APPLICATION

USUS is the society devoted to users of the p-System and UCSD Pascal. Its goal is to promote and influence the development of the p-System and to help users learn more about their systems.

USUS provides both formal and informal opportunities for members to communicate with and learn from each other. Its semiannual national meetings and quarterly newsletters feature technical presentations and discussions as well as news about the p-System and its derivatives. Electronic mail bulletin boards put you in touch with a member network that can provide up-to-the-minute information, and special interest groups zero in on specific problem areas. USUS also supports a Software Exchange Library from which members can obtain software source code for a nominal reproduction charge.

Developed to facilitate software portability, the p-System is the most widely used, machine-independent software system. Pascal was its prinicpal language, but now other languages such as Assembler, Pilot, Lisp, Modula-2, FORTRAN and BASIC are available.

USUS stands for the UCSD p-System User's Society and is pronounced "use us." It is nonprofit and vendor independent.

If you're a p-System user, then USUS is for you. USUS links you with a community of users who share your interests. The following benefits are available to USUS members:

SOFTWARE EXCHANGE LIBRARY
- Tools, games, aides
- Pascal source
- Nominally priced

INFORMATIVE NATIONAL MEETINGS
- Tutorials
- Technical presentations
- Special interest group meetings
- Low-cost software library access
- Hardware/software demonstrations
- Query "major vendors"

HELP VIA ELECTRONIC COMMUNICATIONS
- CompuServe/MUSUS SIG
  - Bulletin board
  - Data bases
  - Software library
- Telemail

USEFUL QUARTERLY NEWSLETTER
- Technical articles and updates
- SIG reports
- Software vendor directory
- Library catalog listings
- Organizational news

ACTIVIST SPECIAL INTEREST GROUPS

TECHNICAL ARCHIVE

Appendix F

USUS MEMBERSHIP APPLICATION

I am applying for $25 individual membership _____
$500 organization membership _____
$__ air mail service surcharge _____

Rates are for 12 months and cover surface mailing of the newsletter. (If you reside outside North America, air mail service is available for a surcharge. It is as follows: $5.00 annually for those in the Caribbean, Central America and Columbia and Venezuela; $10.00 annually for those in South America, Europe, Turkey and North Africa; and $15.00 for all others.) Check or money order should be drawn on a U.S. bank or U.S. office.

Name/Title_____

Affiliation_____

Address_____

_____Country____

Phone (___)____-_____ TWX/Telex_____

Option: Do not print my phone number in USUS rosters ___
Option: Print only my name and country in USUS rosters ___
Option: Do not release my name on mailing lists ___

Computer System:

___Z-80 ___8080 ___PDP/LSI-11 ___6502/Apple ___6800
___6809 ___9900 ___8086/8088 ___Z8000 ___68000
___MicroEngine ___IBM PC Other_____

I am interested in the following Committees/Special Interest Groups (SIGs):

___Advanced System Editor SIG
___Apple SIG
___Application Developer's SIG
___Communications SIG
___DEC SIG
___File Access SIG
___Graphics SIG
___IBM Display Writer SIG
___IBM PC SIG
___Meetings Committee
___NEC Advanced PC SIG
___Publications Committee
___Sage SIG
___Software Exchange Library
___Technical Issues Committee
___Texas Instruments SIG
___UCSD Pascal Compatibility SIG

Mail completed application with check or money order payable to USUS and drawn on a U.S. bank or U.S. office, to Secretary, USUS, P.O. Box 1148, La Jolla, CA 92038, USA.

APPENDIX G
SOFTWARE PROBLEM REPORT

If you encounter any problems with the p-System software, you should report them to SofTech Microsystems or your p-System supplier using the form in this appendix.

Reporting problems is a practice that benefits everyone. Customers can learn that the problem has already been solved, and what the fix is, or that it was previously unknown, and that steps will be taken to fix it in future versions. Software authors benefit from the reports—not everyone is familiar with all the problems which users discover, nor all the applications for which the p-System might be used. New uses lead to new problems, which lead, in turn, to new improvements.

Some users try to fix problems on their own, without consulting their supplier. We ask that you do report problems, even if you think they may already be known (it isn't necessarily true), or if you have found some private solution (the solution you find may be something your supplier would like to know).

We do ask that you be aware of the difference between a software problem and a design suggestion. Some people will inevitably object to things that are intended features of the system. There is nothing wrong with that—the design process itself involves debate and compromise. If you have a suggestion, please report it—only through feedback can the system improve. The p-System documents attempt to describe the p-System that is sent out. If there are discrepancies between the documents and your software, you should consider them to be software (or documentation) problems. If the manuals accurately describe the situation you object to, then report your dissatisfaction, but realize that the way the system operates is already known.

When you report a problem, the more information you provide, the better. These are the things that should be specifically stated:

Environment:

1. What part of the system was running?

2. What version of p-System were you using?

3. What processor do you use?

Actions:

1. What were you trying to do?

2. What were you doing immediately before the problem appeared?

3. What exactly happened that was a problem? In what order did the events related to the problem occur?

Reactions:

1. Have you figured out a workaround?

2. How seriously does the problem affect your work?

3. Have you had this problem before (even transiently)?

If you possibly can, you might include a listing with your report. Often a listing will be needed to understand a problem.

Remember that debugging is the slowest part of any software development, so don't expect problems to disappear overnight. Nonetheless, we fully appreciate the time you take to fill out a useful report. Your concern for the p-System is what keeps it maturing.

Details of who to contact for support assistance should be included with the system you receive. If you receive your p-System through a supplier other than SofTech Microsystems, then you should always contact that supplier directly, unless you have been specifically instructed otherwise. You may contact SofTech Microsystems through mail or by phone:

SofTech Microsystems, Inc.
16885 West Bernard Drive, Suite 300
San Diego, California 92127

(619) 451-1230

Please copy and fill out the appropriate portions of the following three pages if you wish to report a problem.

p-SYSTEM PROBLEM REPORT

Your name:
Address:
Date:
Phone:
Registration #:

Is a reply necessary? (Y/N)

Impact (see below for definition):

none, mild, moderate, severe, lethal

Is this a report of a:

_____ Software problem?
_____ Document problem?
_____ What document?
_____ Page Number(s)
_____ Design suggestion?

What portion of the p-System is affected?

_____ Assembler (LSI-11, 8080, Z80, Z8, 6800, 6502, 6809, 9900, 8086, 68000)
_____ Compiler (Basic, Fortran, Pascal)
_____ Debugger
_____ Editor
_____ Filer
_____ Linker
_____ Long Integers
_____ Operating System
_____ Optional Product _____
_____ p-machine emulator (interpreter)
_____ Utilities _____
_____ Other _____

What is its version number? (for example, IV.12A)

What processor do you use, and who manufactures your system or terminal?

Please describe your problem. If possible, tell us how we may duplicate it, and whether you have found a workaround. If there was an error message, please include the entire message.

Please send any listings or additional pages which may help to analyze your problems.

Impact Definitions

1. None — Implies that the problem is harmless or merely cosmetic in nature.

2. Mild — Implies actions that could confuse or mislead the user, but don't create unexpected results.

3. Moderate — Implies that unexpected results do occur and the system may need to be reinitialized or rebooted to recover.

4. Severe — Implies that a significant amount of work is lost (for example, loss of one disk file or the result of an editing session).

5. Lethal — Implies a system crash that may purge all files from a disk.

APPENDIX H
p-SYSTEM GLOSSARY

Adaptable System	A variation of the p-System that allows you to write the low-level device interface code which handles the peripherals on a specific computer. Once this installation process is done, the p-System can be used on the new computer.
Anchor	In the Screen-Oriented Editor, the position of the cursor when D(elete is invoked. When the cursor is moved away from this position, text disappears. When the cursor is moved toward this position, text reappears.
Application Program	A computer program that meets specific needs of a personal computer user. Examples include a payroll program or an oil well supervision program.
Assembler	A program that translates human-readable assembly language into machine code.

Associate Time	The time taken by the Version IV operating system to find and stitch together the units referenced by a program. This stitching together must occur before the program can begin execution.
Back File	A backup file for text files that is identified by the suffix .BACK; for example, FILENAME.BACK.
Backup	The operation of making an extra copy of important information (usually on a storage volume, in this book). Also, the extra copy that results from this operation.
Bad Block	A 512-byte area on a storage volume that is somehow damaged. The result is that information can't be stored or retrieved from there.
Bad File	An immobile file used to prevent the use of bad blocks on a disk. A bad file is identified by the suffix .BAD; for example, BAD.00120.BAD.

BASIC	A popular high-level programming language that is supported in the p-System.
BIOS	Basic Input/Output Subsystem; that portion of a p-machine emulator that is specific to a particular brand of computer.
Bit	The minimum unit of storage on most computers. A bit is either "on" or "off."
Block	The 512-byte unit of storage and retrieval that is used with p-System storage volumes.
Block-Structured Device	Referred to in this book as "Storage Volume." Earlier p-System documentation, and many p-System prompts and error messages still use "block-structured device," or "blocked device," when referring to storage volumes.
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."

Bug	A defect in a program that causes it not to operate as intended.
Byte	A unit of computer storage. Usually has the capacity to store 8 bits of information, or a number in the range 0 through 255.
Chaining	See "Program Chaining."
Client	A program or unit which uses another unit.
Code File	A file that contains the compiled or assembled version of a program or program segment. Usually identified by the suffix .CODE; for example, FILENAME.CODE.
Code Segment	The smallest component of a p-System program that can be moved into (or removed from) main memory during the running of the program.
Communication Volume	A p-System I/O device that doesn't store information on a long-term basis; for example, the console or the printer.

Compilation Unit	A unit (as represented in any of the three p-System languages) or a program. The smallest module that a language allows to be compiled separately.
Compiled Listing	The source lines of a program, annotated by the compiler with details of the results of compilation, including sizes of statements, sizes of data areas, and other information.
Compiler	A program that translates the human-readable source text of a program into p-machine–executable p-code.
Copy Buffer	In the editor, a storage area in which text can be temporarily stored after it has been deleted from the work-space or while it is being copied from one place to another in the work-space.
Cursor	An indicator that highlights a particular point on a display screen. In many situations, characters typed at the keyboard appear on the screen at the location of the cursor.

Data Entry Prompt	See "Prompt."
Data File	A file that contains arbitrary user data. No particular internal structure is assumed. No special file name suffix is required, but .DATA is often used.
Declare	To establish the name and type of an identifier used in a computer program. Some languages (Pascal, for instance) require that all identifiers be declared before they are used.
Decode	A utility used to inspect the contents of code files.
Default	A state or action which will take effect unless an explicit action is taken to choose another possibility. For instance, in S(et E(nvironment in the editor, there are many options that can be set. All of them have default settings which determine the operation of the editor until they are changed.

Default Disk	The volume where the p-System looks for a file unless the file specification explicitly indicates another volume.
Delimiter	A "fence" that marks the boundaries of a sequence of characters. In the editor, for instance, delimiters enclose the target string sought by F(ind. These delimiter characters can't be letters or numbers, but they can be any of the special characters, such as "&" or "/".
Device	Peripheral equipment accessible to the p-System. There are two varieties: storage and communication. Originally, and sometimes still, a device was referred to as a "unit." This usage has been changed to avoid confusion with the UCSD Pascal language construct of the same name.
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:.

Direction Indicator	In the Screen-Oriented Editor or EDVANCE, the flag at the upper-left corner of the screen that indicates the assumed direction for various editor operations.
Directory	An area on a storage volume that contains "housekeeping" information (such as names and locations) about the files on the volume.
Directory Listing	A human-readable list, usually on the console, of the files on a given storage volume, along with miscellaneous information about each file.
Editor	A p-System program that is used to examine, create and modify text files.
EDVANCE	The Advanced Editor. EDVANCE incorporates a wide range of enhancements over the p-System Screen-Oriented Editor.
Execute	To give control of the p-System to a program (usually via the X(ecute activity).

Execution Error	An error detected by the p-System during the execution of a program. When such an error is detected, a message is produced on the console. The message includes error coordinates indicating the program section that was executing when the error occurred. Usually the program must be canceled and the p-System reinitialized.
Execution Option String	A sequence of execution option statements, usually entered in response to the X(ecute prompt. Individual execution options can affect a variety of aspects of p-System operation, such as the prefix volume, the source of input, and so on.
Extended Memory	A facility available on some p-Systems that allows programs to use up to 64K bytes of main memory for data, plus another 64K bytes for program segments.

File	A named collection of information on a storage volume. Also (less frequently), a stream of information transmitted through a communication volume.
File Specification	A description of a source for input or a destination for output in the p-System. A file specification has three major components, all of which are optional: the Volume ID, the File Name, and the Size Specification.
File Suffix	One of several special endings for file names. The file suffix usually indicates the file type. The standard file suffixes are .TEXT, .CODE, .SVOL, .BACK, .DATA, .BAD, and .FOTO.
Floating Point Number	See "Real Number."
Format	To prepare a disk for use with the p-System. This involves writing addresses and other control information on the disk. Any user information previously stored on the disk is destroyed by this operation.

FORTRAN-77	A popular high level programming language supported in the p-System.
Foto File	A file that contains graphic images for use by Turtlegraphics. The name of the file has the suffix .FOTO; for example, PICTURE.FOTO.
Fragmented	The condition of a p-System storage volume when the total unused space on it is spread among many small areas. The size of the largest file that can be stored on a fragmented volume is the size of the largest single area.
Identifier	The name of an object in a programming language such as Pascal.
I/O	Input and output.
I/O Error	An error detected by the p-System during an input or output operation. For example, a disk write will fail if the disk has been inappropriately removed from its drive. An I/O error is one kind of execution error.

I/O Redirection	A feature that allows the p-System's input to come from some place other than the keyboard. Also, output for the p-System can be sent to some place other than the screen.
I/O Result	A number indicating the success or failure of a p-System I/O operation. If this number is zero, the operation was a success; otherwise, the number identifies the problem that occurred during the I/O operation.
Instruction Set	The fundamental operations that a microprocessor is capable of performing. Different kinds of microprocessors usually have different instruction sets.
Integer Number	A whole number (without a fractional part).
Interpreter	See "p-machine emulator."
KSAM	Keyed sequential access method; a file management facility available for the p-System.

Library	A code file that contains one or more units which can be used by programs or other units.
Library Text File	A text file containing a list of library file names. When a program is invoked, the libraries listed in the current library text file are searched for any units needed by the program.
Library Utility	The p-System library management facility. It is used to inspect, modify, and create libraries and other code files.
Linker	A p-System program that combines assembled code files with each other or with a compiled code file. Also called a "link editor."
Long Integer	A language feature of UCSD Pascal that supports integer arithmetic with up to 36 decimal digits of precision.
Marker	A named, invisible flag on a particular location within a text file.

Menu	A list of available activities that is displayed on the screen by the operating system and many p-System programs. An activity can be selected from a menu with a single keystroke.
Microprocessor	A miniaturized computer. Provides the computational power for most personal computers. Executes the instructions of the software running in the personal computer.
Module	A component of some larger structure with the attribute that it can be handled separately from the rest of the structure in some sense. A UCSD Pascal unit is a module of a program.
Mount	To cause a subsidiary volume to be accessible to the p-System.
Multitasking	The execution of two or more tasks concurrently within a single UCSD Pascal program.
Native Code	Machine level code that is produced by the native code generator as the translation of a section of p-code.

Native Code Generator	A program that translates portions of an executable p-code file into native code. The resulting code file always contains a combination of p-code and n-code.
n-code	See "Native Code."
Nonblock-Structured Device	Referred to in this book as "Communication Volume." Earlier p-System documentation, and many p-System prompts and error messages still use "nonblock-structured device," or "unblocked device," when referring to communication volumes.
Object Code	The machine-readable representation of a computer program.
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	A widely used high level language. UCSD Pascal, an extended version of this language, is the principal programming language in the p-System.
p-code	Psuedo-code: p-machine code generated by the p-System compilers and executed by the p-machine emulators.
p-machine	An idealized pseudo-computer optimized for high-level language execution on small host machines; the foundation of the p-System's portability.
p-machine emulator	The part of the p-System that allows a host microcomputer to imitate the operation of the p-machine. It is implemented in the assembly language of the host computer.
PME	See "p-machine emulator."

Portability	The ability to move executable code between dissimilar microcomputers without recompilation or other change. This is possible in the p-System because programs are compiled into p-code that can be executed on any computer on which the p-System has been installed.
Prefix Disk	See "Default Disk."
Print Spooler	A facility for printing text files concurrently with other activities in the p-System (particularly text editing).
Procedure	A named subprogram that handles part of the job of a larger program or unit.
Program	A set of detailed instructions that direct a computer in the performance of a specific task. Also, the process of creating such a set of instructions.
Program Chaining	Causing the automatic execution of one program from another program.

Prompt	A request (by a p-System program) for information from the p-System user; the user is expected to enter the information at the keyboard, followed by <return>.
p-System	A portable microcomputer software environment for execution and development of applications programs.
RAM	Random Access Memory. A computer's main memory.
RAM Disk	A logical storage volume maintained in main memory. It can generally be used for the same purposes as a conventional disk volume (including storage of files), but the information it contains is usually lost when the computer is turned off.
Real Number	A number that can have a fractional part, such as "5.67982".
Reboot	To start up the p-System again. To "rebootstrap."
Redirect	See "I/O Redirection."
Root Volume	See "System Disk."

Run-time Software	p-System software that is needed to run programs.
Screen-Oriented Editor	The principal text editing tool of the p-System. It is optimized for use with display consoles, rather than printing consoles.
Script File	A file containing characters representing the keystrokes that you would type during a session with the p-System. When p-System input is redirected to this script file, those keystrokes are read as if they were coming from the keyboard, and the session is recreated.
Segment	See "Code Segment."
Source Text	The human-readable form of a computer program. (Also referred to as "source code.")
Special Character	A visible character that isn't a number (0 through 9) and not a letter (A through Z). Examples of special characters include "*", "/", "(", and "@".

Special Key	A keyboard key that has a particular meaning to the p-System other than representing an ordinary visible character. Example: the <return> key.
Storage Volume	An input/output device that can store information written to it, for retrieval at a later time. Usually some sort of a disk, but can be an area of main memory, as well. (See "RAM Disk.")
Subsidiary Volume	A file on a storage volume that contains its own volume structure with a directory and files. This subsidiary volume becomes accessible to the p-System when it is "mounted." The subsidiary volume facility of p-System Version IV.1 supports a two-level file heirarchy.
Substitute String	The character pattern that is to take the place of instances of the target string which are found by the R(eplace activity in the Screen-Oriented Editor.
.SVOL File	A file identified by the suffix .SVOL that contains a subsidiary volume; for example, NAME.SVOL.

Syntax	The rules governing the structure of a program written in a computer programming language.
Syntax Error	A place in a computer program where the rules of the programming language are violated.
System Disk	The disk from which the p-System was bootstrapped. It contains the operating system software. Also known as "root" or "boot" disk. All three of these adjectives also occur with "volume" instead of "disk."
System Files	The disk files which contain the main components of the p-System.
Target String	The character pattern sought by the F(ind and R(eplace activities in the Screen-Oriented Editor.
Text File	A file that contains user-readable information (as opposed to machine code); usually identified by one of the suffixes .TEXT or .BACK.

Turtlegraphics	A package of routines that creates and manipulates images on a graphic display.
Type Ahead	A capability of a p-System implementation to store keystrokes that are typed before the p-System is ready to process them.
UCSD	University of California at San Diego. Site of the original development work on the p-System.
UCSD Pascal	A programming language, an extended version of the language Pascal.
UCSD Pascal System	The original name of the p-System.
Unblocked Volume	See "Nonblock-Structured Volume."
Unit	A package of routines and associated data structures written in a p-System programming language (usually UCSD Pascal). The facilities implemented by the unit (or a subset of them) can be used by programs or by other units.

Universal Medium	A 5-1/4" diskette format that is accessible to many types of small computers. It facilitates the distribution of p-System based personal computer application programs.
Utilities	Programs that assist in various areas of p-System use such as developing programs, maintaining files, printing files, and so forth.
Volume	A logical entity representing a p-System peripheral device. There are two categories of volumes: storage volumes (such as a disk) and communicaton volumes (such as the console or the printer).
Volume ID	Short for "Volume Identifier." The designation of a particular volume; for instance, its name or device number.
Wild Cards	Special symbols in file names that allow a group of files to be represented by a single file name.

Window	In the Screen-Oriented Editor, the portion of the display screen that is used to show a section of the work-space being edited.
Work File	Special file(s) that are automatically processed by major p-System components, including the editors and compilers. This automatic handling is particularly convenient during the development of small programs.
Work-Space	Text kept in main memory by a p-System Editor during the editing process. Also called the "buffer."

Write-Protect	Mark a storage volume in some way so that an error is reported if the p-System attempts to write information onto the volume. (Reading is allowed, but writing isn't.) Used to protect valuable data from accidental erasure. The physical mechanism used to signal write-protection of a volume varies with the storage medium used. For instance, 5-1/4" diskettes have a different convention than 8" diskettes. Check the documentation for your computer to find out how to write-protect the media that you use.
XenoFile	A utility package that allows you to access disks that contain data formatted for the CP/M operating system.
YALOE	Yet Another Line-Oriented Editor; the p-System editor used with printing terminals rather than with display terminals.

INDEX

- B -

- C -

- D -

- E -

- F -

- H -

- I -

- K -

- L -

- M -

- N -

- O -

- P -

- Q -

- R -

- S -

- T -

- U -

- V -

- W -

- Y -