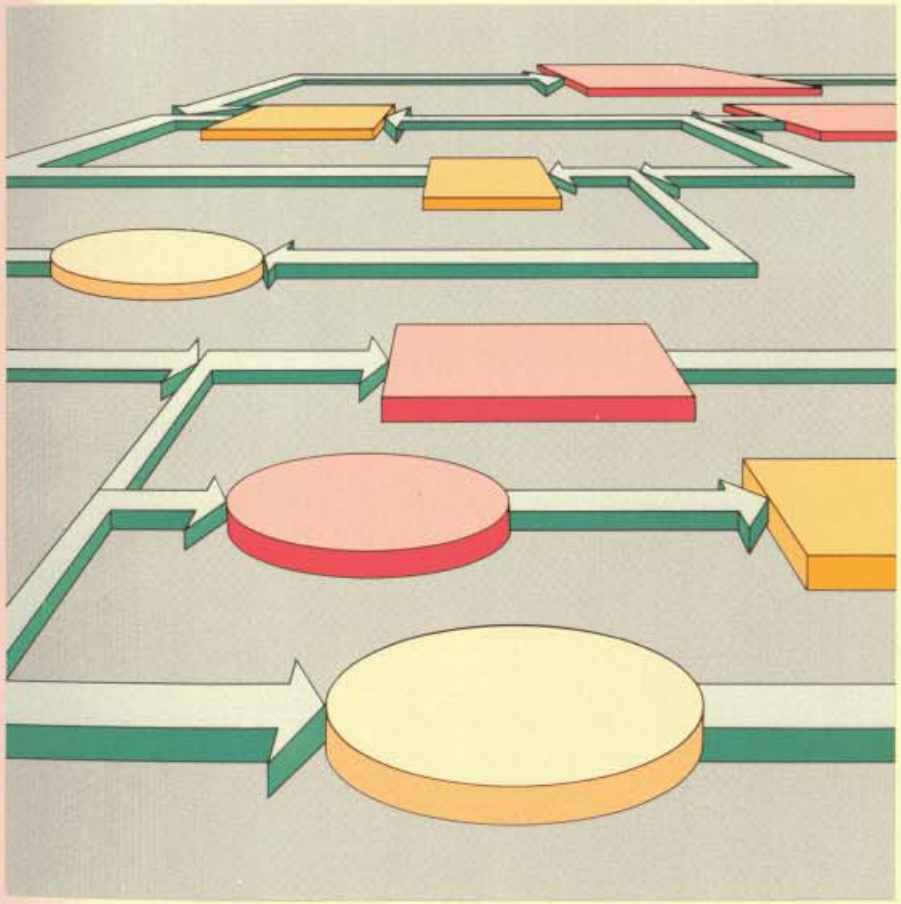
apple computer

20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010
TLX 171-576

030-0602-A

Apple II

Apple Pascal 1.2 Update Manual



Customer Satisfaction

If you discover physical defects in the manuals distributed with an Apple product or in the media on which a software product is distributed, Apple will replace the documentation or media at no charge to you during the 90-day period after you purchased the product.

In addition, if Apple releases a corrective update to a software product during the 90-day period after you purchased the software, Apple will replace the applicable diskettes and documentation with the revised version at no charge to you during the six months after the date of purchase.

In some countries the replacement period may be different; check with your authorized Apple dealer. Return any item to be replaced with proof of purchase to Apple or an authorized Apple dealer.

Limitation on Warranties and Liability

Even though Apple has tested the software described in this manual and reviewed its contents, neither Apple nor its software suppliers make any warranty or representation, either express or implied, with respect to this manual or to the software described in this manual, their quality, performance, merchantability, or fitness for any particular purpose. As a result, this software and manual are sold "as is," and you the purchaser are assuming the entire risk as to their quality and performance. In no event will Apple or its software suppliers be liable for direct, indirect, incidental, or consequential damages resulting from any defect in the software or manual, even if they have been advised of the possibility of such damages. In particular, they shall have no liability for any programs or data stored in or used with Apple products, including the costs of recovering or reproducing these programs or data. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

Copyright

This manual and the software (computer programs) described in it are copyrighted by Apple or by Apple's software suppliers, with all rights reserved. Under the copyright laws, this manual or the programs may not be copied, in whole or part, without the written consent of Apple, except in the normal use of the software or to make a backup copy. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased (with all backup copies) may be sold, given or loaned to another person. Under the law, copying includes translating into another language.

You may use the software on any computer owned by you but extra copies cannot be made for this purpose. For some products, a multi-use license may be purchased to allow the software to be used on more than one computer owned by the purchaser, including a shared-disk system. (Contact your authorized Apple dealer for information on multi-use licenses.)

Product Revisions

Apple cannot guarantee that you will receive notice of a revision to the software described in this manual, even if you have returned a registration card received with the product. You should periodically check with your authorized Apple Dealer.

© 1981, 1982, 1983 by Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010

Apple and the Apple logo are registered trademarks of Apple Computer, Inc.

Simultaneously published in the U.S.A. and Canada. All rights reserved.

Apple II

Apple Pascal 1.2 Update Manual

ACKNOWLEDGMENTS

The Apple® Pascal system incorporates UCSD Pascal™ and Apple extensions for graphics and other functions. UCSD Pascal was developed largely by the Institute for Information Science at the University of California at San Diego, under the direction of Kenneth L. Bowles.

"UCSD Pascal" is a trademark of The Regents of the University of California. Use thereof in conjunction with any goods and services is authorized by specific license only and is an indication that the associated product or service has met quality assurance standards prescribed by the University. Any unauthorized use thereof is contrary to the laws of the State of California.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES vii

CHAPTER 1

INTRODUCTION TO PASCAL 1.2 1

3 Symbols Used in This Manual
4 What Is Pascal 1.2?
4 Who Needs Pascal 1.2?
5 How to Use This Manual
6 About Your Pascal 1.2 Software
6 The Pascal 1.2 Disks
6 Making Copies for Backup
7 Mixing Pascal 1.1 and Pascal 1.2
8 Running Version-1.1-Compiled Programs Under 1.2
8 Using Your New Software Right Away
9 Starting Up a One-Drive System
9 Starting Up a Two-Drive System

CHAPTER 2

NEW PASCAL FEATURES 11

13 Features for All Apple II Computers
13 Improved Disk-Formatting Program
14 A Changed Two-Stage Startup or "Boot"
14 A New Line on the Pascal Startup Screen
14 If You Do Not Put Back the Startup Disk
15 The Percent Prefix
16 Accessing Files During Program Execution
17 Chaining to Other Programs During Execution
17 A New Swapping Option
19 Additional Block Volume Units
19 Error Message: Too Many Program Segments
19 Control Characters Not Echoed to the Screen
19 Hand-Control Buttons and the SHIFT Key
20 The CTRL-] Function
20 Features for the Apple IIe Computer
20 Lowercase and Uppercase Both Available
20 Four Cursor Keys Now Available
20 Keystroke Functions Not Used
21 OPEN-APPLE, SOLID-APPLE, and SHIFT Keys

21 Apple IIe's With Foreign Keyboards
21 User Break During Program Execution
22 Special MISCINFO Files and How to Use Them
24 Steps for 40-Column Apple II Users
25 Steps for 80-Column Apple II Users
25 Steps for 40-Column Apple IIe Users

CHAPTER 3

THE PASCAL 128K SYSTEM 27

29 The Extended 80-Column Text Card
30 Making a 128K System Startup Disk
32 128K System User Error Messages
33 128K System Memory Organization
38 Memory Organization Features
38 Managing Auxiliary Memory
40 Additional Segments
40 How to Use the New Libraries
40 Important Definitions
42 Comparing Libraries Under the 64K and 128K Systems
45 Making a Library Name File
46 Using the Library Name File
46 Using One Library File With Two Programs
47 Using Several Library Files With One Program
49 Using the Percent Prefix in a Library Name File
50 How the System Searches Libraries

CHAPTER 4

TIPS FOR PROGRAMMERS 53

55 A New Swapping Procedure for Programs
57 A New Function Checks a Remote Device
59 Four New Screen-Control Characters
59 The "Ignore External Terminal" Flag: Apple II and Apple IIe
60 The OPEN-APPLE, SOLID-APPLE, and SHIFT Key Controls
61 The High-Bit Test for the OPEN-APPLE Key
64 The UNITSTATUS Test for All Three Keys
66 Three Special Identification Flags
67 Flag to Check the Computer Type
68 Flag to Check the Pascal System Version
68 Flag to Check the Interpreter Version
69 Two Important Pointer Locations
70 New Values for the Up-Arrow and Down-Arrow Keys
71 Reading the Up-Cursor and Down-Cursor Values
72 Changes to the SEEK and PUT Procedures
73 Two Features No Longer Operative

APPENDIX A

BUG FIXES IN PASCAL 1.2 75

77 Compiler Bugs
78 Assembler Bugs
79 Linker Bugs
79 LIBRARY.CODE Bug
79 LIBMAP.CODE Bug
79 SEEK/PUT Bugs
80 80 Input/Output Bugs
80 Turtlegraphics Bugs
81 Miscellaneous Execution-Time Bugs

APPENDIX B

THE FILES ON THE PASCAL 1.2 DISKS 83

APPENDIX C

ERROR MESSAGES 85

87 Compiler Error Messages
90 New Assembler Error Message

APPENDIX D

ACTIVATING THE SHIFT-KEY MOD 91

INDEX 93

UCSD PASCAL SYSTEM USER'S SOCIETY (USUS) 97

USUS MEMBERSHIP APPLICATION 99

LIST OF FIGURES AND TABLES

CHAPTER 1: INTRODUCTION TO PASCAL 1.2

7 Table 1-1. Summary of Files on the Pascal 1.2 Disks

CHAPTER 2: NEW PASCAL FEATURES

18 Table 2-1. Swapping Options Available at the System Level
23 Table 2-2. Options for Customizing Screen Width

CHAPTER 3: THE PASCAL 128K SYSTEM

34 Figure 3-1. The Pascal 64K System: Apple II and IIe
36 Figure 3-2. The Pascal 128K System: Apple IIe
39 Table 3-1. Pointers for the Pascal 128K System (Hexadecimal Values)
41 Figure 3-3. The File Pathname
45 Table 3-2. Pascal Library Options: 64K and 128K Systems

CHAPTER 4: TIPS FOR PROGRAMMERS

57 Table 4-1. Swapping Options You Can Set From Programs
59 Table 4-2. New Screen-Control Characters
61 Table 4-3. Testing for Use of OPEN-APPLE, SOLID-APPLE, and SHIFT Keys
67 Table 4-4. Hardware Identification Bit Settings
69 Table 4-5. Version Flags Set at Location -16606 ($BF22 Hex)
70 Table 4-6. Two Pascal Pointers

APPENDIX D: ACTIVATING THE SHIFT-KEY MOD

92 Table D-1. SHIFT-Key Mod Character Translations
92 Table D-2. Effects of an Activated and Inactivated SHIFT-Key Mod

CHAPTER 1
INTRODUCTION TO PASCAL 1.2

3 SYMBOLS USED IN THIS MANUAL
4 WHAT IS PASCAL 1.2?
4 WHO NEEDS PASCAL 1.2?
5 HOW TO USE THIS MANUAL
6 ABOUT YOUR PASCAL 1.2 SOFTWARE
6 The Pascal 1.2 Disks
6 Making Copies for Backup
7 Mixing Pascal 1.1 and Pascal 1.2
8 Running Version-1.1-Compiled Programs Under 1.2
8 Using Your New Software Right Away
9 Starting Up a One-Drive System
9 Starting Up a Two-Drive System

CHAPTER 1

INTRODUCTION TO PASCAL 1.2

In this chapter, you will learn about the main features of Pascal 1.2 and how to use this manual.

SYMBOLS USED IN THIS MANUAL

This manual uses three symbols to call your attention to important points:

This means the adjacent paragraph contains information especially useful to you--a "helping hand."

This tells you to be alert. The adjacent indented paragraph describes an unusual aspect of Pascal 1.2.

This stop sign is a warning. Pay attention! The adjacent indented paragraph describes an action that could be hazardous to the program or files you are using, or to your computer hardware.

WHAT IS PASCAL 1.2?

Apple II Pascal combines a language and an operating system. You can use it on an Apple IIe computer or on an Apple II or II Plus computer that has at least 48K memory capacity and an Apple Language Card.

From this point on in this manual, the term "Apple II" refers to both the Apple II and the Apple II Plus, as distinguished from the Apple IIe.

Pascal 1.2 is an improved version of Pascal 1.1. The basic program design and the way the user interacts with it have not changed. The improvements consist of new features, corrections of bugs, a 128K system, and various modifications supporting the use of the Apple IIe computer.

Pascal 1.2 consists of four system disks labeled

APPLEØ:
APPLE1:
APPLE2:
APPLE3:

and a set of two manuals in addition to this 1.2 Update manual:

- Apple Pascal Language Reference Manual (with Addendum)
- Apple Pascal Operating System Reference Manual (with Addendum)

WHO NEEDS PASCAL 1.2?

You need Pascal 1.2 (software and manuals)

- If you will be using Apple II Pascal for the first time and want to start out with an up-to-date system;
- If you have been using Pascal 1.1 or 1.Ø and want to take advantage of the several improvements found in version 1.2;
- If you plan to use Pascal on the Apple IIe computer and want the Pascal features that particularly support the Apple IIe;
- If you have an Apple IIe with the Extended 8Ø-Column Text Card and want to take advantage of the additional memory capacity and features available with the Pascal 1.2 128K system.

HOW TO USE THIS MANUAL

You should first page through this manual to become familiar with its topics and the kinds of reference aids available in the Appendixes.

If you are a new user of Apple II Pascal, you should learn the Pascal system by studying the set of original Pascal manuals and addenda, and by practicing the use of various components, such as the Pascal Editor, Filer, Compiler, and so on. Then you should go to Chapter 2 of this manual, "New Pascal Features," which discusses in detail the improvements to Pascal and certain options available within the system.

If you are a practiced user of Apple II Pascal, you should scan the contents of this manual for what might be helpful to you, particularly Chapter 2, "New Pascal Features," and Chapter 4, "Tips for Programmers."

If you are an Apple IIe user, you should read the description of Pascal modifications supporting the Apple IIe in Chapter 2, "New Pascal Features."

If you use the 4Ø-column screen width on an Apple II or IIe, or the 8Ø-column screen width on an Apple II, you should read the section in Chapter 2 called "Special MISCINFO Files and How to Use Them" before using your Pascal 1.2 system.

If you are now using or plan to use the Apple IIe with the Extended 8Ø-Column Text Card, read Chapter 3, "The Pascal 128K System."

Appendix A is a list of bugs in Pascal 1.1 that have been fixed in Pascal 1.2.

Appendix B gives a complete list of the files on the Pascal 1.2 disks.

Appendix C presents an updated list of all Compiler error messages, as well as the one new Assembler error message.

Appendix D explains how to activate the SHIFT-key modification in the event that this hardware change has been made to your computer and you want your Pascal system to use it.

ABOUT YOUR PASCAL 1.2 SOFTWARE

Your time will be well spent if you take a few minutes now to get acquainted with your Pascal software before starting up the system.

THE PASCAL 1.2 DISKS

Table 1-1 is a summary of the contents of each Pascal 1.2 disk. You may arrange these files to suit your special text-editing or program-development needs. (See Appendix B for an itemized list of the files on the four Pascal 1.2 disks. You will find a table that describes the individual system files in Appendix D of the Apple Pascal Operating System Reference Manual.)

As a precaution, however, you should not rearrange the files on the original disks or on the backup copies you will make. Rather, you should prepare a special, customized disk, transferring those files to it that you want together.

MAKING COPIES FOR BACKUP

Before going on, make a copy of each Pascal 1.2 system disk for your everyday use, storing the originals as backups in case of disk damage or unusual wear. See Appendix D in the Apple Pascal Language Reference Manual for directions on making backup disks.

The Disk	Its General Contents and Purpose
APPLEØ:	Contains all the files needed to edit and run Pascal programs, especially on a one-drive system; it includes SYSTEM.COMPILER, but not SYSTEM.APPLE, which is needed to start up the system. This is the second of two disks used for a two-stage startup on a one-drive system.
APPLE1:	Contains all the files you need to edit text and to start up the system. In conjunction with the APPLE2: disk, it is used to Compile or Run your text.
APPLE2:	Contains the Compiler, Linker and Assembler, as well as certain other program-development tools.
APPLE3:	Contains SYSTEM.APPLE, the Formatter program, a few demonstration programs for the general user, and the files named 128K.PASCAL and 128K.APPLE, which are special versions of system files needed for using the additional memory available with the Apple IIe Extended 8Ø-Column Text Card. It also contains three specialized MISCINFO files: II4Ø.MISCINFO, IIE4Ø.MISCINFO, and II8Ø.MISCINFO. This is the first of two disks used for a two-stage startup.

Table 1-1. Summary of Files on the Pascal 1.2 Disks

MIXING PASCAL 1.1 AND PASCAL 1.2

You should not mix any system files from the two versions of Apple II Pascal. The two versions are incompatible because essentially all of the files were changed in the updating from Pascal 1.1 to 1.2.

The Pascal 1.2 operating system (in the SYSTEM.PASCAL file) and the other components of the Pascal system (the Filer, Editor, Compiler, Assembler, Linker, and others) must work together as a unit. The 1.2 operating system--any 1.2 component, in fact--should not be run with a 1.1 version of any other Pascal component. The operating system will check for this condition at execution time and notify you of an incorrect version of a Pascal system component.

RUNNING VERSION-1.1-COMPILED PROGRAMS UNDER 1.2

In general, version 1.2 is compatible with application programs that were compiled under version 1.1, allowing you to run programs under Pascal 1.2 that were designed to run under Pascal 1.1. In special circumstances, however, you might have to make one or both of the following changes:

- You may have to upgrade the original SYSTEM.LIBRARY file that supported the application program and resides on the program disk. The reason is that in the Pascal 1.2 SYSTEM.LIBRARY, these units have been changed or are affected by changes in the operating system:

  PASCALIO
  CHAINSTUFF
  LONGINTIO
  TURTLEGRAPHICS

  Consequently, if you have on a program disk a Pascal 1.1 SYSTEM.LIBRARY file with any of these units, you will need to replace such units with their counterparts from the Pascal 1.2 SYSTEM.LIBRARY. You change units in a SYSTEM.LIBRARY file by means of the Pascal Librarian program explained in Chapter 8 of the Apple Pascal Operating System Reference Manual.

- You may have to change the program to get the correct values for the up-cursor and down-cursor keys, if the program uses these because programs hard-coded to check for the Pascal 1.1 up-cursor and down-cursor keyboard values will not work properly if run under Pascal 1.2. You will need to change such programs to obtain the new values from the Pascal 1.2 SYSTEM.MISCINFO file at load time.

USING YOUR NEW SOFTWARE RIGHT AWAY

If you are using an Apple IIe with an 80-column card and know how to start up and use the Pascal language and operating system, you can use Pascal 1.2 right away. However, if you use an Apple II or an Apple IIe without an 80-column card, you should change SYSTEM.MISCINFO files according to the directions given in the third section of Chapter 2, "Special MISCINFO Files and How to Use Them."

STARTING UP A ONE-DRIVE SYSTEM

To start up Pascal 1.2 on a one-drive system, follow these steps:

1. Insert APPLE3: in the drive.

2. If the computer's power is off, turn it on. If it is already on, press CONTROL-RESET (on an Apple II) or CONTROL-OPEN-APPLE-RESET (on an Apple IIe).

   In some cases, such as after running a copy-protected program, you may have to turn the power off, then on.

3. After the message

   "Insert boot disk with SYSTEM.PASCAL on it, then press RETURN"

   appears on the screen, insert APPLE0: in the drive and press RETURN.

Once you have started the system, you can restart it by selecting Halt or Initialize from the command line (with APPLE0: in the drive).

STARTING UP A TWO-DRIVE SYSTEM

To start up Pascal 1.2 on a two-drive system, follow these steps:

1. Insert APPLE1: in drive 1.

2. If the computer's power is off, turn it on. If it is already on, press CONTROL-RESET (on an Apple II) or CONTROL-OPEN-APPLE-RESET (on an Apple IIe).

   In some cases, such as after running a copy-protected program, you may have to turn the power off, then on.

Once you have started system, you can restart it by selecting Halt or Initialize from the command line (with APPLE1: in drive 1).

CHAPTER 2

NEW PASCAL FEATURES

13 FEATURES FOR ALL APPLE II COMPUTERS
13 Improved Disk-Formatting Program
14 A Changed Two-Stage Startup or "Boot"
14 A New Line on the Pascal Startup Screen
14 If You Do Not Put Back the Startup Disk
15 The Percent Prefix
16 Accessing Files During Program Execution
17 Chaining to Other Programs During Execution
17 A New Swapping Option
19 Additional Block Volume Units
19 Error Message: Too Many Program Segments
19 Control Characters Not Echoed to the Screen
19 Hand-Control Buttons and the SHIFT Key
20 The CTRL-] Function
20 FEATURES FOR THE APPLE IIE COMPUTER
20 Lowercase and Uppercase Both Available
20 Four Cursor Keys Now Available
20 Keystroke Functions Not Used
21 OPEN-APPLE, SOLID-APPLE, and SHIFT Keys
21 Apple IIe's With Foreign Keyboards
21 User Break During Program Execution
22 SPECIAL MISCINFO FILES AND HOW TO USE THEM
24 Steps for 40-Column Apple II Users
25 Steps for 80-Column Apple II Users
25 Steps for 40-Column Apple IIe Users

CHAPTER 2

NEW PASCAL FEATURES

Whether you use a standard Apple II, Apple II Plus, or Apple IIe computer, the first section of this chapter tells you about new Pascal features you may use. If you use an Apple IIe, you should read the second section as well. Also read the third section, on special MISCINFO files, if you use an Apple II or an Apple IIe with a 40-column screen.

FEATURES FOR ALL APPLE II COMPUTERS

The following changes are important for all users of Pascal 1.2 on any Apple II or IIe computer.

IMPROVED DISK-FORMATTING PROGRAM

The program that prepares a new disk before it can be used--the Pascal Formatter--has improved, more meaningful error messages. The same two disk files are used as before:

FORMATTER.CODE - found on disk APPLE3:; used in any drive.

FORMATTER.DATA - incorrectly called "FORMATTER.TEXT" in the Apple Pascal Operating System Reference Manual; found on disk APPLE3:; used in any drive.

After you type X FORMATTER.CODE, you see this revised screen message:

```
APPLE PASCAL DISK FORMATTER PROGRAM [1.2]

FORMAT WHICH DISK (4, 5, 9..12) ?
```

The Formatter error messages have been revised and increased in number to help you better understand why the program is having trouble formatting your disk. These are the error messages that you might see displayed:

- Disk is write protected
- Unable to format diskette
- Drive speed is too slow
- Drive speed is too fast

For instructions on using the Formatter, see the Apple Pascal Operating System Reference Manual, Chapter 8.

A CHANGED TWO-STAGE STARTUP OR "BOOT"

If you use a two-stage startup procedure to begin running your Pascal 1.2 system, you will find an important change in the prompt that comes on your screen after you start up your first system disk. You will be directed to insert your second startup disk (one containing SYSTEM.PASCAL) and press RETURN. (Under Pascal 1.1, the prompt asked you to press RESET. Now, if you press RESET by itself, nothing will happen.) For information on how to start up ("boot") your system, see the end of Chapter 1 of this manual and see Chapter 2 of the Apple Pascal Operating System Reference Manual.

A NEW LINE ON THE PASCAL STARTUP SCREEN

The Pascal startup screen now displays a new line that specifies whether the Pascal interpreter and operating system you are using are 64K or 128K. Every time you start up Pascal 1.2, the first screen display to appear will include either the words

```
Pascal System Size is 64K
```

or the words

```
Pascal System Size is 128K
```

(The Pascal 128K system is discussed in Chapter 3 of this manual.)

IF YOU DO NOT PUT BACK THE STARTUP DISK

If the system returns to the command line and you have not put the startup disk back in drive 1, you now see an expanded reminder on the screen:

```
Put in <boot diskette>:
then press RETURN
```


Under Pascal 1.1, only the first line of the message appears, and drive 1 spins continuously until you insert the correct startup disk.

THE PERCENT PREFIX

Pascal 1.2 gives you a tool that makes your program independent of volume names. You can now use the percent character (%) as a prefix to a filename to mean "the same volume name as the executing program." For example, if the program

```
MYFILE:MIX.CODE
```

is currently being executed, the percent prefix can be used to represent the volume name

```
MYFILE:
```

during the execution of this program and until another program is executed.

Instead of giving the volume name and filenames of files used by the program, such as

```
MYFILE:DATA1
MYFILE:DATA2
```

your program can now simply specify them by attaching the percent prefix to their filenames:

```
%DATA1
%DATA2
```

The percent prefix allows you to write an application program that can call files without hard-coding volume names into it. The application can be on any volume in the system as long as the files used by the program reside on the same volume. Moreover, the user can move the program and its related files to another volume in the system--flexible (sometimes called "floppy") disk or rigid disk--without changing the program.

To use the percent prefix, you first place the files, such as the data files just mentioned, in the same volume as the executing program, and then you use the percent prefix, whenever you need it, as a substitute for the volume name. This capability frees you from having to know and use the volume name of the program file (and of the program's library and data files).

When you execute a program, the percent prefix is set as soon as the system has determined that the volume name and filename are valid and refer to an actual file. (The volume that contains this file must be on line.) The prefix is not set to another volume name while the current program is executing, but when you execute another application

program, or a system program such as the Pascal Filer, Editor, or Compiler, then the percent prefix is set by the system to another volume name, which is that of the new program.

Although you can use the percent prefix at the system level--for example, with the List or Transfer command of the Filer--note that it has three basic uses within a program:

- accessing files during program execution (discussed later in this chapter)
- chaining to other programs during execution (discussed later in this chapter)
- naming files in a Library Name File (128K system only, discussed in Chapter 3)

Accessing Files During Program Execution

Most application programs require the use of numerous files (like data files, output files, temporary files, and so forth) during execution. These files usually reside in the same volume as the main program. Using the percent prefix, you can specify these files in the main program without having to know their volume name. For example, if the program MIX.CODE uses the files DATA1 and DATA2, you would want to group the set of programs in the same volume:

```
MYFILE:              {a volume}
  MIX.CODE           {an executable program}
  DATA1              {a data file}
  DATA2              {a data file}
```

Then in the source code for program MIX.CODE, you can specify the two data files using the percent prefix in these strings:

```
%DATA1
%DATA2
```

Here are two examples of source code showing possible uses of the percent prefix:

```
RESET(A_FILE, '%DATA1');
REWRITE(B_FILE, '%DATA2');
```

Thus you do not have to specify the actual volume name (in this case, MYFILE:). You are free to place this set of files in any volume, with any name, as long as they all reside in the same volume and as long as that volume is on line at the time of program execution.

Chaining to Other Programs During Execution

When your program uses chaining, you can use the percent prefix to specify the volume name of the program to be chained to. For example, if you want the set of programs

```
MASTERPLAN:          {a volume}
  PARAMS.CODE        {an executable program}
  BUDGET.CODE        {an executable program}
  GOALS.CODE         {an executable program}
  FORECST.CODE       {an executable program}
```

to be executed in the order GOALS.CODE --> PARAMS.CODE --> BUDGET.CODE --> FORECST.CODE, you use these calls to the SETCHAIN procedure:

- In GOALS.CODE use the procedure call

```
SETCHAIN('%PARAMS');
```

- In PARAMS.CODE use the procedure call

```
SETCHAIN('%BUDGET');
```

- In BUDGET.CODE use the procedure call

```
SETCHAIN('%FORECST');
```

By using the percent prefix when specifying the next file to be chained to, you avoid having to know the file's volume name. To start running the programs in the chain, you execute MYFILE:GOALS. Again, all that is necessary is that you place the files on line and in the same volume.

Chaining to a program during execution is explained in the Apple Pascal Language Reference Manual Addendum.

A NEW SWAPPING OPTION

Version 1.2 of the Pascal operating system makes available 822 additional bytes of memory that can be used for any activity that needs more system memory.

Application program writers should not depend on this extra memory being available in the future because Apple Computer, Inc. has reserved it for future use. No more memory is guaranteed than that available under Pascal 1.1.

To obtain this additional memory, you will need to use the revised Swap command accessed from the command line. The new prompt screen for the Swap command gives three swapping options. The first two correspond to

the old "toggle" option that permitted you to turn swapping on or off and that made available 2,234 extra bytes of memory. The third option provides the additional memory. (Table 2-1 lists the three swapping options.)

The new swapping option provides more space by moving the procedures GET and PUT from disk to main memory only as they are needed by your program. To do this, set swapping to "2" from the prompt screen that appears up when you type "S" for Swap from the command line.

Note the warning that using GET or PUT to disk will be slow if you select Swapping option 2, since these routines will have to be loaded repeatedly.

Swapping Option	Selection Code	System Action	Total Memory Gain
Swap is off	Ø	Swapping set to OFF. Set automatically at startup, or boot, time. Or set by typing "Ø" after typing "S" from the command line.	--
First level Swap is on	1	First-level swapping set to ON in order to gain space in main memory. Set by typing "1" after typing "S" from the command line.	2,262 bytes
Second level Swap is on	2	Second-level swapping set to ON in order to gain even more space in main memory. Incudes everything swapped at first level and adds 822 more bytes. Set by typing "2" after typing "S" from the command line.	3,Ø84 bytes

Table 2-1. Swapping Options Available at the System Level

For a description of how to use these swapping options from a program when chaining to another program, see the section "A New Swapping Procedure For Programs" in Chapter 4.

ADDITIONAL BLOCK VOLUME UNITS

Eight new units, numbers 13 through 2Ø, have been added to the original available block units, numbers 4, 5, 9..12. The operating system treats the new units the same as it did the original ones when, for example, it scans units as it looks for a particular system program, such as the Pascal Compiler. (The new units are useful only for attached block devices, such as large-volume, rigid-disk drives.) To use these units for a rigid-disk drive, you would need its device driver and the SYSTEM.ATTACH program.

If you use only flexible-disk drives, you will not be able to use these new units because the Apple II and IIe have no slots corresponding to the new unit numbers where you could install additional flexible-disk drives.

ERROR MESSAGE: TOO MANY PROGRAM SEGMENTS

A new error message appears on the screen if you attempt to run a program that has segment numbers larger than 31 or uses intrinsic units with segment numbers larger than 31. The new error message:

Specified code file must be run under the 128K Pascal system.

In this case, the code file must be run on the Pascal 128K system. Only users of an Apple IIe with an Extended 8Ø-Column Text Card are able to convert to the Pascal 128K system with its larger available memory, enhanced library capabilities, and additional 32 segments. (See Chapter 3.)

CONTROL CHARACTERS NOT ECHOED TO THE SCREEN

Pascal 1.2 will not echo or write to the screen any control characters typed on the keyboard except CONTROL-M or CONTROL-G.

HAND-CONTROL BUTTONS AND THE SHIFT KEY

You can design and run programs that test the positions of the hand-control buttons Ø and 1 (or the OPEN-APPLE and SOLID-APPLE keys on an Apple IIe), pressed by the user in response to a program prompt. The SHIFT key may be similarly used, provided the SHIFT-key modification (discussed in Appendix D) is made first. The section "OPEN-APPLE, SOLID-APPLE, and SHIFT Key Controls" in Chapter 4 tells how to test the buttons.

THE CTRL-) FUNCTION

This is a reminder, not a new function. Note that on an Apple II, a CONTROL-] function is achieved from the keyboard by pressing CONTROL-SHIFT-M , an action necessary because the right bracket itself, "]", is produced by pressing SHIFT-M .

FEATURES FOR THE APPLE IIE COMPUTER

Several keyboard and control code changes are important to users of Pascal 1.2 on an Apple IIe. You will want to take special note of them if you are accustomed to using Pascal 1.1.

LOWERCASE AND UPPERCASE BOTH AVAILABLE

Lowercase characters, as well as uppercase, are directly available on the Apple IIe keyboard: uppercase characters are produced using the SHIFT or CAPS LOCK key. (The "SHIFT-key modification" for using the SHIFT key to shift between uppercase and lowercase characters is not necessary on the Apple IIe.)

You may have to press CAPS LOCK to run certain applications programmed only for uppercase characters.

FOUR CURSOR KEYS NOW AVAILABLE

The Apple IIe has two additional cursor keys--up and down--as well as the left-cursor and right-cursor keys available on the Apple II. Pascal 1.2 uses these keys to move the cursor if you use the correct MISCINFO file.

KEYSTROKE FUNCTIONS NOT USED

If you have been using Pascal on an Apple II, please note that Pascal on the IIe ignores several keystroke functions that remain in use on the Apple II. When Pascal 1.2 is used on an Apple IIe, it does not use

- CONTROL-E to shift between uppercase and lowercase characters and turn inverse video on;
- CONTROL-W to force the keyboard into uppercase for the next character typed and turn inverse video on;
- CONTROL-R to turn inverse video on but not change the keyboard from uppercase;

- CONTROL-T to turn inverse video off and force the keyboard into uppercase;
- CONTROL-K to produce the left bracket character;
- Other character translations produced by the SHIFT-key modification, where

^	becomes	N
@	"	P
]	"	M
CNTRL-@	"	@
CNTRL-^	"	^
CNTRL-]	"	]

- CONTROL-O and CONTROL-L for up-cursor and down-cursor action.

Note the warning earlier in this chapter that you might have to change the up-cursor and down-cursor code values for Pascal 1.1 programs that check for these values, in order to run these programs under Pascal 1.2. See Chapter 4 for an explanation of the changes to the control codes for up-cursor and down-cursor actions.

In addition, Pascal 1.2 does not use the DELETE key to delete anything, but simply sends the ASCII DEL character (code 7F) to the calling program if requested. The use of this key is left to the software developer; for suggestions, see the Apple IIe Design Guidelines.

OPEN-APPLE, SOLID-APPLE, AND SHIFT KEYS

You can use the Apple IIe OPEN-APPLE, SOLID-APPLE, and SHIFT keys for special function characters, for game controls, or for performing special reset and self-test cycles. See Chapter 4, the section "OPEN-APPLE, SOLID-APPLE, and SHIFT Key Controls," on how to determine from your program when one of these keys is pressed.

APPLE IIE'S WITH FOREIGN KEYBOARDS

If you are using Pascal on an Apple IIe with a foreign keyboard, Pascal 1.2 automatically selects the language character set built into your system.

USER BREAK DURING PROGRAM EXECUTION

On an Apple IIe, you interrupt a program during execution by pressing CONTROL-SHIFT-2 (CONTROL-@). On an Apple II using Pascal, you interrupt a program by pressing CTRL-SHIFT-P (CTRL-@). The ASCII code for this control function remains the same as before.

SPECIAL MISCINFO FILES AND HOW TO USE THEM

Your APPLE3: disk contains three special MISCINFO files, one of which you should use if you use a 40-column or an 80-column screen with an Apple II, or a 40-column screen with an Apple IIe. If you fit any of these three cases, you can learn here how to replace the standard SYSTEM.MISCINFO file on your APPLE0: and APPLE1: disks with either the II40.MISCINFO, the II80.MISCINFO, or the IIE40.MISCINFO file. (If you use an 80-column screen on an Apple IIe, you should use the standard SYSTEM.MISCINFO file supplied on the APPLE0: and APPLE1: disks.)

The cursor-move keys work on any Apple II without MISCINFO file modification unless you use an 80-column screen. In this case, you need to transfer a copy of the file II80.MISCINFO from the APPLE3: disk to the APPLE0: disk, to the APPLE1: disk, and to any other startup disk you use--in each case changing its name to SYSTEM.MISCINFO. If you plan to use Pascal 1.2 on an Apple II or an Apple IIe in 40-column mode, you will get the best results by copying either the special II40.MISCINFO file or the special IIE40.MISCINFO file from the APPLE3: disk to the APPLE0: disk, to the APPLE1: disk, and to any other startup disk you use. Table 2-2 shows the column-width setup for each machine and which MISCINFO file to use.

The II40.MISCINFO file and the IIE40.MISCINFO file are identical to the 80-column MISCINFO file for the Apple IIe except that the screen width in the former two is set to 79 columns to ensure abbreviated Pascal prompt lines on the 40-column screen. (A MISCINFO file that sets the screen width to 80 columns ensures that all Pascal prompt lines appear in unabbreviated form.)

In addition, in the II40.MISCINFO file, the "has lower case" control variable is set to False. The II80.MISCINFO file is the same as the 80-column Apple IIe MISCINFO file except the values for up-cursor and down-cursor movement in the former are the same as under Pascal 1.1 (for up, CONTROL-O, and for down, CONTROL-L).

Apple Model	40-column Feature	80-column Feature
Apple II	Built in when shipped.	You must add an 80-column card or an external video terminal with an 80-column option.
	Use the II40.MISCINFO file resident on the APPLE3: disk of Pascal 1.2.	Use the II80.MISCINFO file resident on the APPLE3: disk of Pascal 1.2.
Apple IIe	Built in when shipped.	You must add an 80-column card or an external video terminal with an 80-column option.
	Use the IIE40.MISCINFO file resident on the APPLE 3: disk of Pascal 1.2.	Use the SYSTEM.MISCINFO file resident on the APPLE0: and APPLE1: disks of Pascal 1.2.

Table 2-2. Options for Customizing Screen Width

Read on for the steps you take in moving the appropriate file into place.

Be sure to modify only copies of your Pascal 1.2 system disks, not the originals, which should be stored intact as backups.

STEPS FOR 40-COLUMN APPLE II USERS

Users of a 40-column screen width on an Apple II should follow the following steps:

1. From the Pascal Filer prompt line, select the "T" or Transfer option, which you will use to copy the proper MISCINFO file to your APPLE0: and APPLE1: disks.

2. Be sure the write-enable notch on your APPLE3: disk is covered with a write-protect tab to protect you from accidentally writing over and deleting any Pascal files on that disk. With your APPLE0: disk in the first (the startup) drive, place your copy (not the original) of the Pascal 1.2 APPLE3: disk in your second drive, if you have one, and answer the "Transfer?" prompt by typing APPLE3:II40.MISCINFO and pressing the RETURN key. (If you have only one disk drive, replace the startup disk in your startup drive with APPLE3:, and answer the "Transfer?" prompt by typing APPLE3:II40.MISCINFO and pressing the RETURN key.)

   Before taking the following step, be sure that the APPLE0: and APPLE1: disks to which you are copying do not have a tab covering their write-protect notches.

3. Then two-drive users should answer the "To where?" prompt by typing APPLE0:SYSTEM.MISCINFO and pressing the RETURN key, assuming that the APPLE0: disk is in place in the startup drive. (One-drive users should first replace APPLE3: with APPLE0: in the startup drive and then answer the "To where?" prompt by typing APPLE0:SYSTEM.MISCINFO and pressing the RETURN key. When the system prompts for the "destination" disk, press the SPACE bar as indicated.) Now the system asks if it should delete the original SYSTEM.MISCINFO before copying. Type "Y" for "Yes" in response, because you want to replace that file with the II40.MISCINFO file.

4. Both two-drive and one-drive users should repeat the above procedure to copy the II40.MISCINFO file again, this time from APPLE3: to APPLE1: .

5. If you have other startup disks you use regularly with the 40-column screen width, you can replace their SYSTEM.MISCINFO file with the II40.MISCINFO file by using the same steps just presented.

STEPS FOR 80-COLUMN APPLE II USERS

Users of an 80-column screen with an Apple II should follow the same steps given for 40-column Apple II users in the preceding section, except you should substitute the filename II80.MISCINFO for the name II40.MISCINFO wherever it is used.

STEPS FOR 40-COLUMN APPLE IIE USERS

Users of the 40-column screen with an Apple IIe should follow the same steps given for 40-column Apple II users, except you should substitute the filename IIE40.MISCINFO for the name II40.MISCINFO wherever it is used.

CHAPTER 3

THE PASCAL 128K SYSTEM

29 THE EXTENDED 80-COLUMN TEXT CARD
30 MAKING A 128K SYSTEM STARTUP DISK
32 128K USER ERROR MESSAGES
33 SYSTEM 128K SYSTEM MEMORY ORGANIZATION
38 Memory Organization Features
38 Managing Auxiliary Memory
40 ADDITIONAL SEGMENTS
40 HOW TO USE THE NEW LIBRARIES
40 Important Definitions
42 Comparing Libraries Under the 64K and 128K Systems
45 Making a Library Name File
46 Using the Library Name File
46 Using One Library File With Two Programs
47 Using Several Library Files With One Program
49 Using the Percent Prefix in a Library Name File
50 How the System Searches Libraries

CHAPTER 3

THE PASCAL 128K SYSTEM

If you are using an Apple IIe Extended 8Ø-Column Text Card, you may customize a startup or "boot" disk that will give you more usable memory in addition to the built-in 64K of RAM in the Apple IIe. This is possible because the Extended 8Ø-Column Text Card provides 64K of auxiliary RAM--for a total of 128K with the card on an Apple IIe. The Pascal 1.2 software contains special 128K versions of the files SYSTEM.APPLE and SYSTEM.PASCAL that you can use instead of the standard 64K versions of both.

THE EXTENDED 80-COLUMN TEXT CARD

The Pascal 128K system enables you to use the extra memory provided by the Extended 8Ø-Column Text Card. It therefore gives you more space for your program code and data. The 128K Pascal System

- Allows up to 46K of compiled P-code storage space on the Extended 8Ø-Column Text Card;
- Allows up to 41K for data and assembly-code storage in the Apple IIe's main memory, because P-code is stored on the Card;
- Allows 64 segments instead of the standard 32 segments;
- Provides enhanced library capabilities, including libraries that can be shared by two or more programs;
- Can be used to edit files up to 58 blocks in length, compared with the standard 34-block files;
- Provides Compiler symbol-table space of 18,719 words, compared to the 1,774 words available using the Pascal 64K system;
- Provides Assembler symbol-table space of 18,487 words, compared to the 8,317 words available using the Pascal 64K system;
- Executes Pascal programs developed on the Pascal 1.2 64K system;

- Executes Pascal 1.1 programs, providing the program is otherwise compatible with Pascal 1.2 characteristics;
- Has a processing speed virtually as fast as that of the 64K system and reduces dependence on the slow swapping methods often used to make more system memory available to programs.

MAKING A 128K SYSTEM STARTUP DISK

To put the 128K Pascal System on your Apple IIe, you create a new startup disk by copying two Pascal 1.2 system files to a newly formatted disk, according to the following instructions.

Do not create a 128K startup disk unless you will be using the Apple IIe Extended 80-Column Text Card. The new startup disk will not work on a 64K Apple IIe.

1. Start up your Pascal 1.2 system and then format a new disk, if you do not already have a supply of Pascal-formatted disks, using the Formatter program as explained in Chapter 8 of the Apple Pascal Operating System Reference Manual. If your system has two disk drives, the newly formatted disk should be in the second drive (unit #5:) during the next few steps. One-drive users should not yet insert the newly formatted disk in their drive.
2. Next, from the Pascal Filer prompt line, select the "T," or Transfer option, which you will use to copy the special 128K files to your new startup disk. Now remove your system disk from the startup drive (unit #4:).
3. Before proceeding, check to be sure the write-enable notch on your APPLE3: disk is covered with a tab to prevent accidentally overwriting a system file. Now place your APPLE3: disk in your startup drive (unit #4:) and answer the "Transfer?" prompt by typing APPLE3:128K.APPLE and pressing the RETURN key.

Note that in the next step, you change the name of the file from 128K.APPLE to SYSTEM.APPLE as you copy it to the new startup disk. For the location of the file, you may give the unit device number (#4 or #5, as shown) or the volume name of your newly formatted disk.

4. Two-drive users should now answer the "To where?" prompt by typing #5:SYSTEM.APPLE, and pressing the RETURN key, assuming that your newly formatted startup disk is already in the second drive. (One-drive users should first replace APPLE3: with the newly formatted disk in the drive and then answer the "To where?" prompt by typing #4:SYSTEM.APPLE and pressing the RETURN key. When the system prompts for the "destination" disk, you press the SPACE bar as indicated.)
5. Both two-drive and one-drive users now repeat the preceding procedure to copy the 128K.PASCAL file from APPLE3: to the new startup disk, changing the name of the file as you do so from 128K.PASCAL to SYSTEM.PASCAL.
6. Again using the Transfer procedure, copy the file SYSTEM.MISCINFO from your APPLE0: or APPLE1: disk to the new startup disk, and copy any other files that your Pascal startup disk requires, such as SYSTEM.LIBRARY.
7. Finally, make a backup copy of this new disk following the directions for copying an entire disk given in Chapter 3 of the Apple Pascal Operating System Reference Manual.

If you try to use the 64K SYSTEM.APPLE file and the 128K SYSTEM.PASCAL file (or vice versa) on the same startup disk, your computer system will either "hang" (not start up) or continually restart ("reboot"). In either case, you will not be able to use Pascal until the file mixup is corrected. (This is one reason you should modify a copy of your startup disk, rather than the original.)

Your new 128K Pascal startup disk is now ready to use. You can use your 128K system all the time: you need not shift back and forth between the 128K and the 64K systems, although you may do so by using a startup disk with the original 64K versions of SYSTEM.APPLE and SYSTEM.PASCAL. (The 64K system will work on a 128K Apple IIe, but will ignore the extra memory.)

128K SYSTEM USER ERROR MESSAGES

You might encounter one or more of these error messages when using the 128K system:

- If you try to start up your customized 128K disk on an Apple IIe without the Extended 8Ø-Column Text Card, the message

  Extended 8Ø-Column Card required

  comes up on your screen, and the system stops. At this point, you will have to restart the system using a 64K startup disk (such as APPLE1:, or, in the case of a two-stage startup, first APPLE3: and then APPLEØ:).

- If code overflows the available space in the RAM of the Extended 8Ø-Column Text Card, the execution error message

  Codespace overflow {if the system disk is on line}

  or Exec Error #16

  comes up on your screen. You must restart the Pascal system by pressing CONTROL-RESET.

- If a running program asks for a new segment but there is less than one block of available stack space, the execution error message

  Stack overflow {if the system disk is on line}

  or Exec Error #4

  comes up on your screen. You must restart the Pascal system by pressing CONTROL-RESET.

128K SYSTEM MEMORY ORGANIZATION

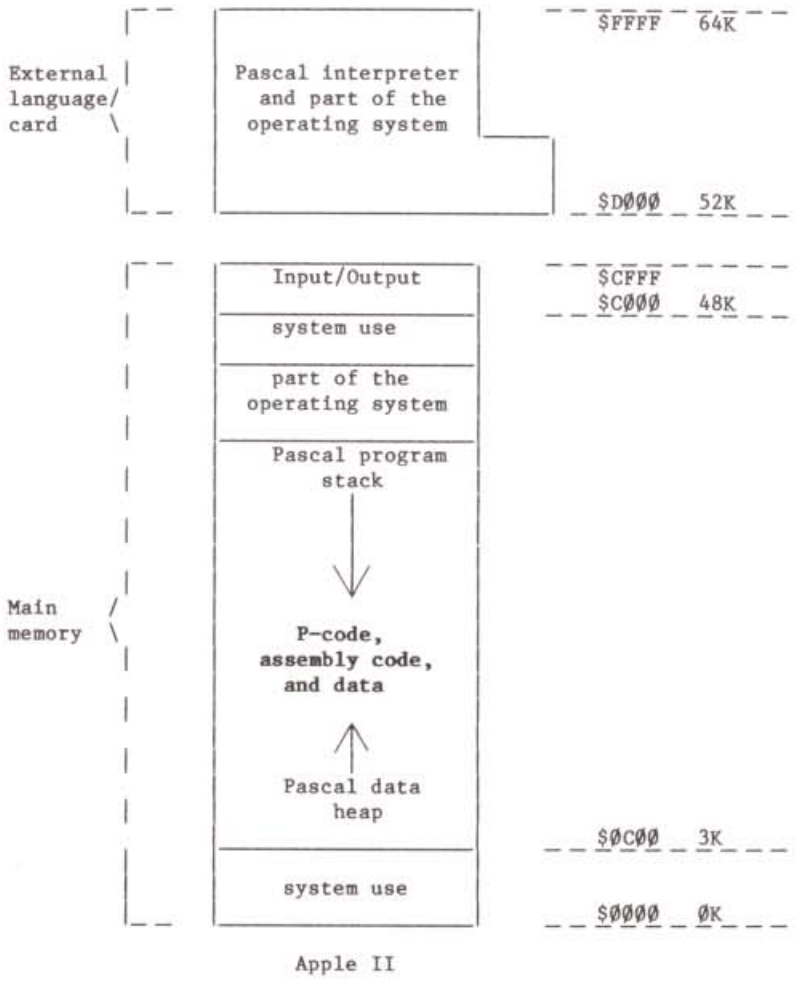
So far you have learned how to make a special startup disk in order to use the Pascal 128K system on an Apple IIe with an Extended 8Ø-Column Text Card. Here you will learn the differences between the ways the 64K and 128K Pascal systems organize memory.

The memory organization of a 64K Pascal system on an Apple II with an Apple Language Card is the same as that on the Apple IIe, which has the 16K RAM of the language card built into its hardware. Both of these 64K systems use corresponding sections of memory for the same functions, as you see in Figure 3-1.

Numbers in figures and text in this manual that are preceded by a dollar sign ($) refer to the hexadecimal number system used in assembly language to refer to addresses in memory. This base-16 system uses the ten digits Ø through 9 and the six letters A through F to represent values from Ø through 15.

The 128K Pascal system and the 64K Pascal systems organize memory in different ways, as you can see by comparing Figures 3-1 and 3-2.

You will find a more detailed memory map of the Pascal 64K system in Appendix B of the Apple Pascal Operating System Reference Manual. For maps and explanations of the auxiliary memory of the Extended 8Ø-Column Text Card, see the Apple IIe Reference Manual, Chapter 4, and the Apple IIe Extended 8Ø-Column Text Card Supplement, Chapter 2.



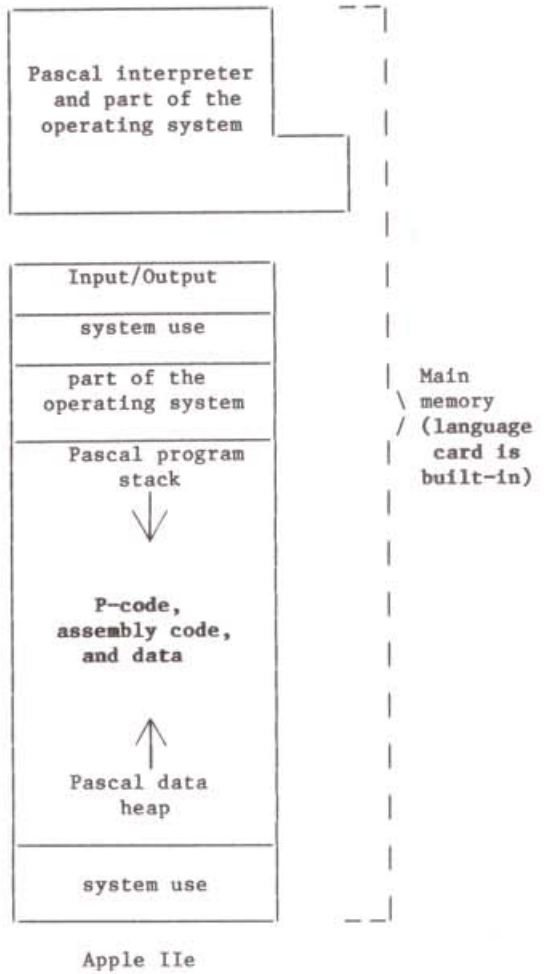


Figure 3-1. The Pascal 64K System: Apple II and IIe

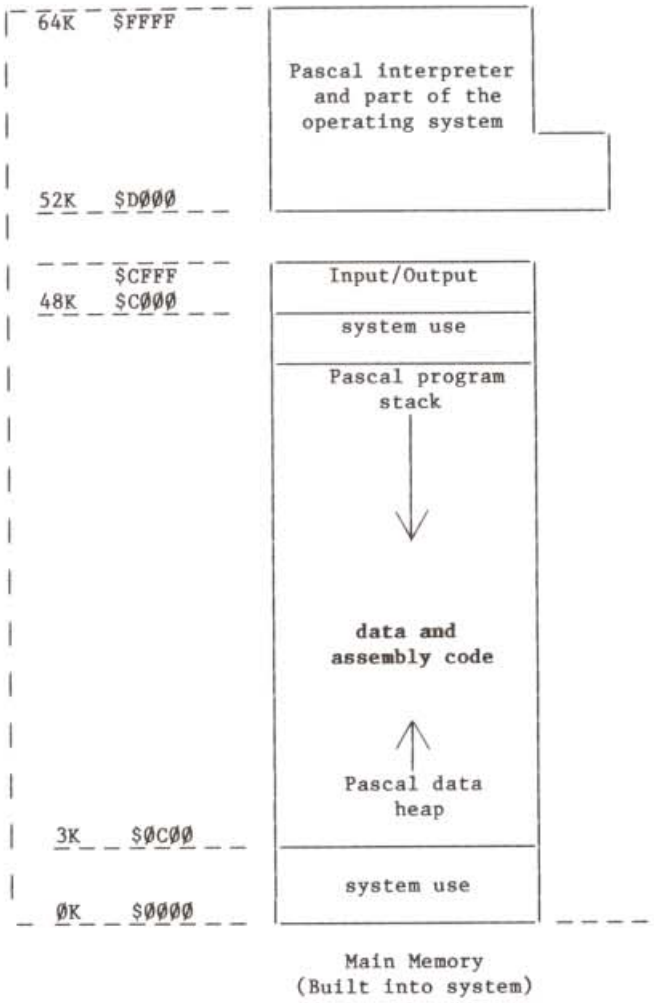


Figure 3-2. The Pascal 128K System: Apple IIe

MEMORY ORGANIZATION FEATURES

These are the most important features of the 128K Pascal System's memory organization:

- The built-in 64K of RAM (the "main memory") stores only assembly code and data.
- All P-code is stored in the "auxiliary memory" space on the Extended 80-Column Text Card.
- Because it is written in P-code, the Pascal operating system has been moved to the auxiliary memory of the card.
- Because they are written in P-code, the Pascal system components (Filer, Compiler, Editor, and so forth) are stored in the auxiliary memory of the card when they are being executed.
- The section of RAM on the Extended 80-Column Text Card that corresponds to the "language card" section in the 64K system is not presently dedicated to a particular function, but is reserved by Apple Computer, Inc. for future system development.
- The 128K Pascal system has enough space to hold the entire Compiler in memory during a compilation. For this reason, it is not necessary to use the {$S+} option when compiling a unit declaration under the 128K system.

These features provide more room to store P-code because it doesn't have to share memory with assembly code and data. In addition, there is more room for assembly code and data in main memory because no P-code is stored there.

Because data and P-code are stored in different sections of memory, using the swapping feature of either the system or the Compiler will not add to the space available for data, but will add to the space available for P-code.

MANAGING AUXILIARY MEMORY

The Pascal 128K system uses two zero-page variables to manage use of the auxiliary memory on the Extended 80-Column Text Card. CODEP points to the lowest used word in the auxiliary memory space. CODELOW contains the lowest permissible value for CODEP; CODELOW defaults to $800. Table 3-1 describes these variables.

Zero-Page Location	Pointer	Description	Permissible Ranges
$60	CODEP	Points to lowest used word in the contiguous 48K of extended RAM space.	$800-$C000
$62	CODELOW	Contains the lowest permissable value for CODEP. Memory below this point is reserved.	Must not be below $800 (default value) or above $C000.

Table 3-1. Pointers for the Pascal 128K System (Hexadecimal Values)

Because CODEP points to the lowest word in the auxiliary memory space, it begins with the value of $C000 and works down until it hits the value CODELOW.

Your program can examine CODEP and CODELOW if it needs to. If your program runs under the 128K system, it cannot change CODEP, but it can change CODELOW if it will use part of the auxiliary memory. For example, to execute a program that uses the 560-dot high-resolution screen, you would change CODELOW to $4000 and then change it back to its original value after the program has run.

If you are using the 64K system on a machine with the Extended 80-Column Text Card, you can use CODEP as a zero-page pointer to the auxiliary memory space on the card. This feature is useful if you are managing this space yourself, rather than using the Pascal 128K system to manage it.

Here are several important reminders about your use of these variables:

1. You must use even numbers when giving values to these variables because they point to words, not bytes.
2. The system does not restore CODELOW or CODEP to their original values after executing your program. Whenever you have changed one of these variables, be sure to put the value back to what it was before your program ends.
3. If your program runs under the 128K system, it can change only CODELOW; CODEP is changed only by the Pascal system.

ADDITIONAL SEGMENTS

The 128K system allows 64 segments, whereas the 64K system allows only 32. This makes it easier to break up a large program into manageable parts. Nevertheless, you can have only 16 segments in a codefile, so that these extra segments will be useful mainly as intrinsic units.

The Compiler, the Linker, LIBRARY.CODE, and LIBMAP.CODE now allow 64 segments, numbered Ø through 63, regardless of which Pascal system (64K or 128K) they are running under. However, if you try to run a program with a segment number greater than 31 under the 64K system, you will get an error. In other words, you can use the 64K system to develop programs that will only run under the 128K system.

Segments 58 through 63 are reserved for use by the Pascal system. They should not be used by an application.

HOW TO USE THE NEW LIBRARIES

This section discusses the extended library file options available only with the Pascal 128K system, shows how to use these options, and also shows how programs may share the same library files. To help you choose your approach to library files as you develop programs, there is a section listing the sequence of steps followed by the system as it searches library files for the intrinsic units required by a program at execution time.

IMPORTANT DEFINITIONS

The library file features available with the 64K system remain unchanged. The Pascal 128K system, however, has increased the number and manageability of library files that you may have for a program. In this discussion of the new library features, you will encounter the phrase "in the same volume as." This phrase means that a file must have the same volume name or unit number as another file. Likewise, the phrase "in a different volume from" means that the volume name or unit number of one file is different from that of another file. You will also encounter the phrase "file pathname." A file pathname is the volume name of the disk, or the unit number of the disk drive, followed by the name of a particular file that resides in the volume. The pathname is the path the system must take to find a given file. See Figure 3-3.

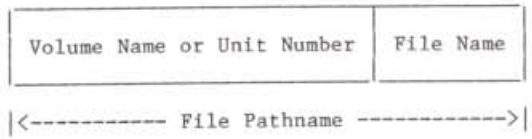


Figure 3-3. The File Pathname

The volume name is whatever name you have given to a particular flexible disk, like MYFILE:, and the unit number is #4:, #5:, or #9:..#2Ø:. The filename is the name of a file in that disk volume, normally including the file-type suffix, like ADDUP.TEXT, ADDUP.CODE, or LIB1.LIB.

In this set of files,

```
MYFILE:              {a volume name}
  LIB1.LIB           {a library file}
  LIB2.LIB           {a library file}
```

the same volume name or unit number heads the file pathname for the two library files

```
MYFILE:LIB1.LIB     { or written as }     #5:LIB1.LIB
MYFILE:LIB2.LIB     { or written as }     #5:LIB2.LIB
```

Notice that in this set of files,

```
NEWSORT:             {a volume name}
  NEW.LIB            {a library file}
  SYSTEM.LIB         {a library file}
```

the file pathname for these two other library files is different from the file pathname for LIB1.LIB and LIB2.LIB:

NEWSORT:NEW.LIB is a different pathname from MYFILE:LIB1.LIB.

And so we say that NEW.LIB is "in a different volume from" LIB1.LIB or LIB2.LIB. Later in this discussion, you will learn how library files in different volumes can be used by a program that is executing.

An understanding of Pascal libraries depends on a clear conception of a few other basic terms used frequently in this discussion, such as "executable code file," "intrinsic unit," and "library file."

An "executable code file" is a file in which all the necessary components are in place: regular units or assembly language code (or both) have been linked, and any required intrinsic units are available in the appropriate library files. An executable code file or a Pascal

library file may be composed of different combinations of compiled and assembled source programs.

In this discussion, we refer mostly to "intrinsic units," occasionally to "regular units." Regular units, by definition, have to be linked with, and thereby inserted in, the executable code file prior to program execution. Intrinsic units, on the other hand, are connected by the system to the executable code file at program execution time. Intrinsic units have two characteristics that are relevant to this discussion: first, they are not restricted to use by only one executable code file; and, second, they must be placed in a library accessible to the system at program execution time in order to be used by the executable code file.

A "library file" is a code file that is not directly executed. Instead, a library file contains one or more compiled intrinsic units used by one or more programs. A USES declaration in the program names the required unit, which is connected by the system at program execution time. Another section of this chapter will explain how the system searches various library files for the intrinsic units required by a particular program.

Two or more library files can be combined into one, using a new name or one of the old names, and units can be moved from one library to another. You may also move units in and out of a copy of the SYSTEM.LIBRARY file that came with your Pascal system. (Chapter 8 of the Apple Pascal Operating System Reference Manual explains how to combine or move library files with the LIBRARY utility program.) The name you give a library file depends on the kind of library file you are using and on its purpose. In general, the suffix .LIB is used to complete the filename.

COMPARING LIBRARIES UNDER THE 64K AND 128K SYSTEMS

Note the differences between the library file system supported by the Pascal 64K system and that supported by the Pascal 128K system. For storing units, the 64K system allows only one library file for each executable program: SYSTEM.LIBRARY. The Pascal 128K system supports SYSTEM.LIBRARY, but also additional libraries called "program libraries."

"SYSTEM.LIBRARY" is a library file that must reside on the system or startup disk in order to be used. It may contain units supplied by Apple Computer, Inc.--the unit called APPLESTUFF, for example--and, if you so choose, additional units that you yourself place in SYSTEM.LIBRARY using the LIBRARY utility program.

A "Program Library File" is a library file that has the same volume name or volume unit number (is "in the same volume as") the executable code file and is given the same name as the executable code file except that its suffix is .LIB rather than .CODE. For example, if an executable code file has this file pathname:

MAIL:SORT.CODE

then the corresponding Program Library File will have this designation:

MAIL:SORT.LIB

A Program Library File, like SYSTEM.LIBRARY, may hold one to sixteen unit segments. Only one Program Library File may be used by a program, although the SYSTEM.LIBRARY file may also be used by the program.

In contrast to the 64K system, the 128K system allows up to six library files (including SYSTEM.LIBRARY) with each executable code file, and also allows multiple programs to share library files. This extension of Pascal libraries is made possible by means of a new kind of file, called a "Library Name File."

A "Library Name File" is a text file you create that contains a list of pathnames of up to five library files that contain intrinsic units you want an executable code file to use. As long as its pathname is correctly given, a library file listed in a Library Name File can be in any volume on line at the start of program execution. The Library Name File uses the same naming convention as a Program Library File: you give it the name of the executable code file, using .LIB as the suffix. (The specific format for a Library Name File is described in the next section of this chapter.)

Note that if you decide to use a Library Name File, you cannot then use a Program Library File because they both would have the same name.

By listing library file pathnames in a Library Name File, you direct the system at the start of execution time to search the files with these pathnames to find any intrinsic units needed by the executable code file. Later in this chapter, you will see how library files in the same volume as the executable code file or in another volume can be listed in a Library Name File and how they can be shared by more than one program.

For an executable code file requiring only a few units, you will find that a Program Library File will take care of your library file needs. For a larger and more complex application--one using a large number of intrinsic units--you should use instead a Library Name File. Using a Program Library File limits you to units residing in the same volume as the executing program. SYSTEM.LIBRARY also has a limited utility for

large applications: it must reside on the system disk, where it takes up valuable space. Furthermore, because you may use a different SYSTEM.LIBRARY in different applications--one you have tailored to fit particular needs--you face the potential conflict of library units having the same name or the same segment number.

These are the advantages of using Library Name Files for your application programs:

- Up to six library files (including SYSTEM.LIBRARY) can be made available to an executable program. As before, each library file can hold up to 16 unit segments, although the maximum number of segments allowed is 64.
- A library file can be shared by two or more executable programs by listing it in separate Library Name Files for each of the executable programs.
- Disk space can be conserved by having only one copy of the same intrinsic unit shared between programs.

Table 3-2 compares the kinds of library options available under the 64K system with those available under the 128K system.

64K System	128K System
Allows one library on line per program: SYSTEM.LIBRARY Must be on system disk Keeps its own name Files can be shared Limit: only one on line	Allows up to six libraries on line per program: PROGRAM LIBRARY FILE Same volume as program Takes name of executable code file and adds .LIB Files cannot be shared Limit: one per program or replace PLF with a LIBRARY NAME FILE Same volume as program Takes name of executable code file and adds .LIB Facilitates library file sharing Limit: one per program Lists pathnames of up to 5 library files LIBRARY FILES Up to 5 usable by a program Any name Can be shared by programs SYSTEM.LIBRARY Must be on system disk Keeps its own name Files can be shared Limit: only one on line

Table 3-2. Pascal Library Options: 64K and 128K Systems

For information on arranging intrinsic units in libraries, see Chapter 5 in the Apple Pascal Language Reference Manual.

MAKING A LIBRARY NAME FILE

A Library Name File is a text file that must conform to a specific text format.

To make a Library Name File, begin a new file in the Pascal Editor. Without leaving the Editor, type "I" to select the Insert option, and make a file using the following format on the left, which is illustrated

by the example on the right:

```
LIBRARY FILES:[RETURN]          LIBRARY FILES:
<pathname>[RETURN]              MAIL:PREP.LIB   ┐
<pathname>[RETURN]              MAIL:FIN.LIB    │  the
<pathname>[RETURN]              MAIL:LIB1.LIB   │  pathnames
<pathname>[RETURN]              MAIL:LIB2.LIB   │  of five
<pathname>[RETURN]              MAIL:LIB3.CODE  ┘  library
$$[RETURN]                      $$                 files
[CONTROL-C]
```

Notes:

1. The "L" in "LIBRARY" must be the first character on the first line in the file. You cannot have any blank lines, spaces, or other characters at the top of the file or between lines. The string "LIBRARY FILES:" may be in uppercase or lowercase. Press the RETURN key after each line, as shown.

2. Below the name "LIBRARY FILES:" and also beginning at the left margin, type on separate lines the pathnames (followed each time by RETURN) for each file you want to designate as a library file. You can have five pathnames or fewer in your file. The system will ignore any pathnames listed after the fifth one.

3. Two dollar signs ($$) make up the last line of the file no matter how many pathnames you use.

4. Press CONTROL-C to leave Insert mode.

After you've made your Library Name File and checked the format carefully, you can type "Q", then "W", to Write it from the Editor to your program disk, giving it the name of the executable code file, but with the .LIB suffix, such as UPDATE.LIB. The following paragraphs tell you in more detail how to select and arrange library files, including those to be shared by using the Library Name Files.

USING THE LIBRARY NAME FILE

This section gives several examples of how to use library files with the Library Name File.

Using One Library With Two Programs

Suppose you have written two short applications, called SORT and UPDATE, each one stored in a separate volume or on a separate flexible disk. Each has to have a set of intrinsic units on line when being executed. Right now the intrinsic units are stored in the library file named PREP.LIB in the same volume (MAIL:) as UPDATE:


```
MAIL:               {a volume}
  UPDATE.CODE       {an executable program}
  PREP.LIB          {a library file}
```

If you wanted either one of the applications to be able to use the intrinsic units contained in PREP.LIB, you would first have to list the pathname of PREP.LIB in a Library Name File in the associated volume, as shown here:

```
MAIL:               {a volume}
  UPDATE.CODE       {an executable program}
  PREP.LIB          {a library file}
  UPDATE.LIB        {a Library Name File . . . . LIBRARY FILES:
                                                  MAIL:PREP.LIB
                                                       .
                                                       .
                                                  $$            }
```

Note that the Library Name File takes the same name (except for the suffix) as the executable code file for the program (UPDATE) that uses it. Also note that for both programs to share the same library file--in this case PREP.LIB--you do not need to place PREP.LIB itself in both volumes. Instead, you leave the file in the volume MAIL: and list its pathname in a Library Name File in the other volume, UTILS:

```
UTILS:              {a volume}
  SORT.CODE         {an executable program}
  SORT.LIB          {a Library Name File . . . . LIBRARY FILES:
                                                  MAIL:PREP.LIB
                                                       .
                                                       .
                                                  $$            }
```

Now PREP.LIB is a shared library file, its intrinsics usable by both programs even though it resides in only one of the two volumes. Of course, the volume MAIL: must be on line when the program SORT is executed so that SORT may have access to the library file PREP.LIB.

Using Several Library Files With One Program

If you have a number of library files in the same volume as the executing program where, for example, the program SEARCH.CODE has the pathname REPORT:SEARCH.CODE, your Library Name File (with the pathname REPORT:SEARCH.LIB) would contain

```
LIBRARY FILES:
REPORT:LIB1.LIB   ┐
REPORT:LIB2.LIB   │   the pathnames
REPORT:LIB3.CODE  ┘     of three
$$                    library files
```

LIB1.LIB, LIB2.LIB, and LIB3.CODE are sample names for library files. (You may use any name for a file containing library units, as we did

for LIB3.CODE, although using the suffix .LIB makes it easier to remember that it is a library of units.)

You could simplify the writing of a Library Name File by setting the Pascal prefix to the name of the volume you are currently using. For example, if, using the Filer, you set the Pascal prefix to the volume name, REPORT:, or, say, #9:, before executing SEARCH.CODE, you could write the Library Name File more simply, like this:

```
LIBRARY FILES:
LIB1.LIB      }      the filenames
LIB2.LIB      }        of three
LIB3.CODE     }      library files
$$
```

The system will attach the prefix to a library filename before opening that file. (To set the Pascal prefix from the Pascal Filer, see Chapter 3 of the Apple Pascal Operating System Reference Manual.)

If you use the Pascal prefix in conjunction with the set of filenames listed in the Library Name File, you must make sure that the prefix is set to produce the correct pathnames so that the program can find its library files when it is executing. If you successively execute programs in different volumes, or programs with library files in different volumes, you will need to change the Pascal prefix before executing each program to ensure that the pathnames for the shared library are correct at execution time. You may find it convenient to rely on setting the prefix during program development, but you would probably not ask a user to set the prefix before running an application program. A more foolproof way would be to use the percent prefix before each filename, as explained in the next section.

A program on one volume can use library files on a different volume. For example, say that you want to use two of the library files in the volume REPORT: for a program called POST.CODE in a second volume, ACCOUNTS:, without physically moving those two files from the original volume (REPORT:). You can do this easily by listing the library files needed by POST.CODE in a separate Library Name File called POST.LIB, to be inserted in the second volume (ACCOUNTS:), using the full pathname to the original volume (REPORT:), as in this example, or using the filename and setting the Pascal prefix to REPORT: before running the program, as in the previous section.

```
ACCOUNTS:           {a volume}
  POST.CODE         {an executable program}
  POST.LIB          {a Library Name File . . . .  LIBRARY FILES:
                                                  REPORT:LIB1.LIB
                                                  REPORT:LIB2.LIB
                                                  $$
```

Note that the library files LIB1.LIB and LIB2.LIB, physically located in the first volume (REPORTS:), are shared by both programs (SEARCH and POST) but that LIB3.CODE is not shared because its pathname is not

listed in the Library Name File in the second volume (ACCOUNTS:). Had it been listed in the Library Name File POST.LIB, then it too would have become a shared library, usable as well by the program POST.CODE, even though the actual library file was physically located in the volume of the program SEARCH.CODE along with the other two library files.

Many possible arrangements of library files are supported by the Pascal 128K system, using different combinations of volumes, programs, files, and disk drives. The examples just mentioned are simply hints to help you get started in developing your own shared libraries. As you can see, you will want to give considerable thought to the overall structure of your application and to the number of disk drives you presently have on line. In particular, you will want to plan the kind of library files appropriate to each program, the files you will designate as shared libraries, and the best arrangement on disk of all the files for a particular application program. The section "How the System Searches Libraries," later in this chapter, gives a brief description of how libraries are searched for the intrinsic code units required by the executing program.

USING THE PERCENT PREFIX IN A LIBRARY NAME FILE

The percent prefix, discussed in Chapter 2, can be used to make a Library Name File independent of its volume name. Because the percent prefix is set to the volume name of the executing code file as soon as that file has been found, you can use the percent prefix in the Library Name File to replace the volume names of the listed library files. If you had this set of files:

```
MYFILE:             {a volume}
  MIX.CODE          {an executable program}
  MIX.LIB           {a Library Name File}
  OLD.LIB           {a library file}
  NEW.LIB           {a library file}
```

and wanted to use the percent prefix, the contents of the Library Name File for MIX.CODE, which is MIX.LIB, would be

```
LIBRARY FILES:
%OLD.LIB
%NEW.LIB
$$
```

Then when you execute MYFILE:MIX.CODE, the system sets the prefix to MYFILE:, opens up the Library Name File MIX.LIB, and reads the pathnames for the two library files OLD.LIB and NEW.LIB. In this case the system expands the pathnames like this:

```
%OLD.LIB    ---->    MYFILE:OLD.LIB
%NEW.LIB    ---->    MYFILE:NEW.LIB
```

The "%" stands for the volume name, MYFILE:, of the program MIX.CODE.

Keep in mind when developing an application that the grouping of related programs and their libraries together in the same volume facilitates the use of the percent prefix to specify library files.

HOW THE SYSTEM SEARCHES LIBRARIES

The following step-by-step description will help you choose the library file approach best suited to the particular application you are developing.

When a program is executed, the system first examines it to determine whether or not it uses any intrinsic units. If it does not, the program is loaded and run. If it does, the system looks at the different types of library files, in the following order, to find the required units:

1. Program Library File
2. Library Name File
3. Library files whose pathnames are listed in a Library Name File
4. SYSTEM.LIBRARY

The system first looks for a file of the same name as the executing program but with the suffix changed from .CODE to .LIB. Then it tries to open the file corresponding to its new name (progname.LIB). If the file exists, the system determines whether it is a code file or a text file. If it finds a code file (the file we call a Program Library File), the system looks in the file for the required intrinsic units. If it finds instead a text file (the file we call a Library Name File), the system collects the pathnames of the library files listed there, and then looks in those files for the required intrinsic units.

If you have set a prefix and the names of the files listed in the Library Name File require a prefix, the system attaches the prefix before searching for the files.

If there are intrinsic units needed that have not been found in a Program Library File or by means of a Library Name File, or if your program has not used either of these libraries at all, the system looks in SYSTEM.LIBRARY. If the missing units are not found in SYSTEM.LIBRARY, or if SYSTEM.LIBRARY is not on the system disk, an error message appears on the screen, and the system returns control to the Pascal command line.

The system searches for the intrinsic units until it finds all of them or until it runs out of library files and gives an error message. If it finds the units before it has looked in all the relevant library files, it stops searching and begins executing the program.

CHAPTER 4
TIPS FOR PROGRAMMERS

55 A NEW SWAPPING PROCEDURE FOR PROGRAMS
57 A NEW FUNCTION CHECKS A REMOTE DEVICE
59 Four New Screen-Control Characters
59 THE "IGNORE EXTERNAL TERMINAL" FLAG: APPLE II AND APPLE IIE
60 THE OPEN-APPLE| SOLID-APPLE| AND SHIFT KEYS
61 The High-Bit Text for the OPEN-APPLE Key
64 The UNITSTATUS Text for All Three Keys
66 THREE SPECIAL IDENTIFICATION FLAGS
67 Flag to Check the Computer Type
68 Flag to Check the Pascal System Version
68 Flag to Check the Interpreter Version
69 TWO IMPORTANT POINTER LOCATIONS
70 NEW VALUES FOR THE UP-ARROW AND DOWN-ARROW KEYS
71 Reading the Up-Cursor and Down-Cursor Values
72 CHANGES TO THE SEEK AND PUT PROCEDURES
73 TWO FEATURES NO LONGER OPERATIVE

CHAPTER 4

TIPS FOR PROGRAMMERS

These notes on more technical aspects of Pascal 1.2 will be of interest to programmers in general and to application developers in particular.

A NEW SWAPPING PROCEDURE FOR PROGRAMS

Apple Pascal provides Swapping options at the system command level and at the program level that allow you to maximize the amount of memory available for program use. (For how these options are used at the system level, see the section "More Memory With a New Swapping Option" in Chapter 2 of this manual.) Under Pascal 1.1, only one level of swapping could be turned on from a program to ensure more memory space for a program about to be chained to and executed. You used it by calling the built-in procedures SWAPON and SWAPOFF found in the CHAINSTUFF unit of SYSTEM.LIBRARY.

Pascal 1.2 includes SWAPON and SWAPOFF plus a new built-in swapping procedure, SWAPGPON, likewise residing in the CHAINSTUFF unit. You can use this additional level of swapping when chaining to programs that require more memory than provided by SWAPON. SWAPGPON provides approximately 8ØØ more bytes of available memory than SWAPON. Like SWAPON, SWAPGPON is called from a program just before it terminates, in order to turn swapping on at this new level for the next program to be chained to. SWAPOFF will turn off all swapping as in Pascal 1.1.

The new swapping option provides more space by moving the procedures GET and PUT from disk to main memory only as they are needed by your program. For this reason, using GET or PUT for files on block-structured devices will be slow when using this swapping option. READ and WRITE, which use GET and PUT, will also be slow. UNITREAD, UNITWRITE, BLOCKREAD, and BLOCKWRITE will be unaffected.

Application program writers should not depend on the extra memory of SWAPGPON being available in the future. No more memory is guaranteed than that available under Pascal 1.1.

Certain planned enhancements to the Pascal system will reduce the memory available to applications by approximately 800 bytes. The new swapping option will allow programs currently running at the limit of available memory to run under the enhanced system.

None of the three swapping procedures takes any parameters when called from your program. However, to use these procedures, you must place a USES CHAINSTUFF declaration immediately after the program heading, give the SETCHAIN procedure call in a program before it terminates, place the appropriate swapping procedure call before the program termination if the swapping level is to be different for the next program, and make sure the the SYSTEM.LIBRARY file is on line when your program is compiled and executed.

Table 4-1 summarizes the swapping procedure options available under Pascal 1.2.

The first two options, SWAPOFF and SWAPON, are documented in the "Swapping Option" section of the Addendum to the Apple Pascal Operating System Reference Manual and in the "Chaining Programs" section of the Addendum to the Apple Pascal Language Reference Manual.

CHAINSTUFF Procedure	System Action	Total Memory Gain
SWAPOFF	Swapping set to OFF. Set automatically at startup time. Or set by calling SWAPOFF from your program before chaining to the next program.	--
SWAPON	First level swapping set to ON in order to gain space in main memory. Set by calling SWAPON from your program before chaining to the next program.	2,262 bytes
SWAPGPON	Second level swapping set to ON in order to gain additional space in main memory. Incudes everything swapped at level 1 and adds 822 more bytes. Set by calling SWAPGPON from your program before chaining to the next program.	3,084 bytes

Table 4-1. Swapping Options You Can Set From Programs

A NEW FUNCTION CHECKS A REMOTE DEVICE

Your programs can use a new built-in function, REMSTATUS, in the APPLESTUFF unit of SYSTEM.LIBRARY to read characters from or write characters to a remote device connected to slot 2 of your Apple II or IIe and to keep the program from waiting if the device is busy. REMSTATUS returns a value of the type RSTATTYPE, which is declared as

```
TYPE     RSTATTYPE = (RSTATBUSY, RSTATREADY, RSTATOFFLINE)
```

The form for calling REMSTATUS is

```
REMSTATUS (RSCHANNEL);
```

where the parameter RSCHANNEL is declared as

```
TYPE RSCHANNEL = (RSOUTPUT, RSINPUT)
```

RSSTATYPE and RSCHANNEL are predeclared in the system. If you put their type declarations in your program, you will get an error.

RSCHANNEL is given the value RSOUTPUT if the program needs to write, RSINPUT if the program needs to read.

This is the way the function works:

1. If there is neither an Apple Communications Card nor a firmware protocol card in slot 2, the function returns the value RSTATOFFLINE.

2. If there is an Apple Communications Card or a firmware card in slot 2 and RSCHANNEL is RSINPUT, the function returns the value RSTATREADY if a character is waiting to be read; othewise it returns RSTATBUSY.

3. If there is an Apple Communications Card or firmware card in slot 2 and RSCHANNEL is RSOUTPUT, the function returns the value RSTATREADY if the output device is ready to accept a character from the program; otherwise it returns RSTATBUSY.

4. If REMSTATUS(RSOUTPUT) = RSTATBUSY and the program writes to the remote device, the program will wait. Similarly, if REMSTATUS(RSINPUT) = RSTATBUSY and the program reads from the remote device, the program will wait.

The following program statements illustrate how REMSTATUS might be used in a terminal emulator program to read characters from or write them to a remote device:

```
REPEAT
  IF (REMSTATUS (RSINPUT) = RSTATREADY) THEN
    BEGIN
      UNITREAD  (7, BUF[Ø], 1, 12);
      UNITWRITE (1, BUF[Ø], 1, 12)
    END;

  IF KEYPRESS AND (REMSTATUS (RSOUTPUT) = RSTATREADY) THEN
    BEGIN
      UNITREAD  (2, BUF[Ø], 1, 12);
      UNITWRITE (8, BUF[Ø], 1, 12)
    END
UNTIL BUF[Ø] = QUITCHAR;
```

Of course, you will need to place a USES APPLESTUFF declaration after your program heading. Because the function is a built-in one, it need not be declared.

FOUR NEW SCREEN-CONTROL CHARACTERS

With any Apple II or IIe 4Ø-column screen or any Apple IIe 8Ø-column screen, you can use the following screen controls.

Change Desired	Program Statement
Make the cursor visible	WRITE (CHR(5));
Make the cursor invisible	WRITE (CHR(6));
Turn inverse video on	WRITE (CHR(15));
Turn inverse video off	WRITE (CHR(14));

Table 4-2. New Screen-Control Characters

These characters will give unpredictable results with some non-Apple 8Ø-column cards.

THE "IGNORE EXTERNAL TERMINAL" FLAG: APPLE II AND APPLE IIE

Pascal 1.2 includes a system flag--identified as the "ignore external terminal" flag--to help the application developer control which screen-width mode is being used. The flag is supplied on both the development (or standard) version and the various run time versions of Pascal 1.2 used by application developers. But the program to set the flag (called RTSETMODE) is provided only on the run-time versions. Application developers will use the flag primarily to test run-time versions of system applications they wish always to run in 4Ø-column mode.

The flag is located in the directory area of the startup disk. Using RTSETMODE, the developer sets the flag by putting a "1" in block 2, byte 25, bit 3. The flag is read by the interpreter at startup time. (Note that these address numbers are relative to zero. In other words, the designation "block 2" refers to the third block, byte 25 to the twenty-sixth byte, and bit 3 to the fourth bit.)

When the flag is set prior to startup, the 1.2 Pascal system will ignore any 8Ø-column firmware card in the Apple. Setting the flag

causes the display to operate only in 40-column mode, even if the Apple computer has an 80-column card installed. In effect, the flag tells the system to "ignore any external terminal card."

The Apple IIe 80-Column Text Card includes a special 40-column screen mode. Although you could enter this special 40-column mode from a program by writing "CONTROL-Q" to the screen, we urge that you not do so. Pascal does not support the use of this special 40-column mode. Using this mode causes the system to behave unpredictably and could harm your program.

THE OPEN-APPLE, SOLID-APPLE, AND SHIFT KEY CONTROLS

The OPEN-APPLE, SOLID-APPLE, and SHIFT keys on the Apple IIe keyboard have several functions, one of the most useful of which allows the user to send responses to an application. (You will find references to these functions in the Apple IIe Reference Manual and the Apple IIe Owner's Manual.)

Your application can test the same kind of user responses on an Apple II. Instead of pressing the OPEN-APPLE or SOLID-APPLE key, the user would press button 0 or button 1, respectively, on the hand controls.

By using one or other of the following methods, your program can test whether the OPEN-APPLE, SOLID-APPLE, or SHIFT key has been pressed--singly or in some combination--along with a character key. The first method allows you to test only for the OPEN-APPLE key. The second method allows you to test whether any of these three keys has been pressed.

The SHIFT key can be tested only if a prior hardware modification has been made to the Apple II or IIe being used to run a particular program. The following discussion points out certain conditions necessary to achieve meaningful results when the SHIFT key is being tested.

You will see several references here to this hardware modification, popularly known as the "game-paddle mod." More accurately, one should talk about it as the "SHIFT-key mod," as this manual does, because the SHIFT key rather than the game paddle connector is the target of the change, though both are involved. To have this modification made to your Apple II or IIe, see your dealer.

You will learn in the next two sections which one of the two types of test to use in your program to check for the use of the OPEN-APPLE (or button 0), SOLID-APPLE (or button 1), or SHIFT key. Table 4-3 gives you an overview of which test is appropriate to which user activity. (The term "Mod" in the table refers to the "SHIFT key modification" just mentioned.)

	Apple II			Apple IIe		
Name of key	Open Apple	Solid Apple	Shift	Open Apple	Solid Apple	Shift
Hardware required	Connect Hand-Ctrls	Connect Hand-Ctrls	Needs "Mod"	Built-in	Built-in	Needs "Mod"
User key	Button 0	Button 1	SHFT Key	O-A Key	S-A Key	SHFT Key
Type of test to use	High-bit or UNITSTAT	UNITSTAT	UNITSTAT	High-bit or UNITSTAT	UNITSTAT	UNITSTAT

Table 4-3. Testing for Use of OPEN-APPLE, SOLID-APPLE, and SHIFT Keys

THE HIGH-BIT TEST FOR THE OPEN-APPLE KEY

There are two ways to check the OPEN-APPLE key on an Apple IIe or button 0 on an Apple II: (1) the high-bit test and (2) the UNITSTATUS test. The high-bit test checks to see whether the high-order bit of the character read has been set as the result of the user pressing the OPEN-APPLE key along with the character key. An ordinal (ORD) value of 128 or more returned by the READ statement indicates that the value

of the high-order bit (128) has been added to the ASCII code value of the character, signalling that the OPEN-APPLE key has been pressed along with a character key. From a Pascal program, you can use the Pascal built-in KEYBOARD or INPUT file to read whatever character was typed. HIBITEST is a program illustrating this method of checking the OPEN-APPLE key or button Ø.

```
PROGRAM HIBITEST;

VAR   KEY :CHAR;

BEGIN
  REPEAT
    READ (KEYBOARD, KEY);
    WRITE (KEY, ' = ', ORD (KEY), '   ');
    IF ORD(KEY) >= 128 THEN WRITE ('OPEN-APPLE PRESSED');
    WRITELN
  UNTIL KEY = 'Q'
END.
```

Normally, your own HIBITEST procedure would not include the WRITE statement, which here acts as an illustration, but would substitute other statements using the input values from the READ statement.

Note that the high-bit test works only for the OPEN-APPLE key or button Ø; you cannot use this same test for the SOLID-APPLE and SHIFT keys or for buttons 1 or 2. Rather, the UNITSTATUS method, described in the next section, can be used to test these, as well as to test the OPEN-APPLE key or button Ø. Why use the high-bit test at all if the UNITSTATUS test works for all three keys? The answer is that testing the OPEN-APPLE key or button Ø using the high-bit test is a little simpler and faster.

If you are an application developer who does not want the high bit of the character byte to be set by pressing the OPEN-APPLE key or button Ø, you may disable this function if you wish. To do so, use a variant record to put a zero in location $BF11 (decimal -16623). A variant record, sometimes called a "trix" record, allows a program to interpret the same physical data in the variant part of the record as being of different types. This device circumvents the Pascal system's normal restriction against mixing variable types. In the program DISABLE that follows, a reference to the field INT interprets the data as an integer, while a reference to the field PTR interprets the same data as a packed array.


```
PROGRAM DISABLE;

TYPE      PA = PACKED ARRAY [Ø..Ø] OF Ø..255;
        ADDR = RECORD
                 CASE BOOLEAN OF
                   FALSE:  (INT:INTEGER);
                   TRUE:   (PTR:^PA)
                 END;

VAR     A:ADDR;

BEGIN
  A.INT := -16623;                {location $BF11}
  A.PTR^[Ø] := Ø;
END.
```

If you have a program that prompts the user to press OPEN-APPLE or button Ø along with a character key, but you have disabled setting the high-order bit, then the program will have to use the UNITSTATUS test to determine whether the OPEN-APPLE key or button Ø has been pressed.

The program ENABLE shows how to re-enable the high-order bit setting so that HIBITEST can be used.

```
PROGRAM ENABLE;

TYPE      PA = PACKED ARRAY [Ø..Ø] OF Ø..255;
        ADDR = RECORD
                 CASE BOOLEAN OF
                   FALSE:  (INT:INTEGER);
                   TRUE:   (PTR:^PA)
                 END;

VAR     A:ADDR;

BEGIN
  A.INT := -16623;                {location $BF11}
  A.PTR^[Ø] := 128;
END.
```

ENABLE is just like DISABLE, except for the change in this statement

```
A.PTR^[Ø] := 128;
```

which again enables setting the high-order bit.

THE UNITSTATUS TEST FOR ALL THREE KEYS

With this test, your program can determine that

1. a character has been typed;

2. the OPEN-APPLE or SOLID-APPLE key (or both) on an Apple IIe was pressed when a character was typed;

3. button Ø or button 1 (or both) on an Apple II was pressed when a character was typed;

4. the SHIFT key was pressed when a character was typed on an Apple II or an Apple IIe with the "SHIFT key mod" installed.

Several things may cause you to get unexpected results from the UNITSTATUS test:

1. Only the SHIFT key on an Apple II or IIe will set the SHIFT-key flag when pressed. Pressing CAPS-LOCK, CONTROL-E, or CONTROL-W does not set the SHIFT-key flag, even though doing so may change certain other keypress results by activating the previously installed SHIFT-key mod. (See Appendix D in this manual for a discussion of these other changes.)

2. If the Apple II or IIe does not have the SHIFT-key mod, the flag set when the user presses the SHIFT key may produce a random value that has no meaning when the UNITSTATUS test is made.

3. If the hand control is not connected to the Apple II, the flag set when the user presses button Ø or button 1 may produce random values that have no meaning when the UNITSTATUS test is made.

4. If you have an Apple II with the hand control connected and you press button Ø at startup time, the system will automatically disable setting the high-order bit of the character, and so the high-bit test will not work. Your instructions to the user should include a caution on this matter.

UNITSTATUS is a Pascal built-in procedure facilitating input/output operations at the memory address level, just as do the more familiar UNITREAD and UNITWRITE procedures. Here, in this context, your program will use UNITSTATUS to retrieve from the console driver a value representing which of the three keys has been pressed. Your program can also read the keyboard buffer data-byte, as it did in the high-bit test, to get the character the user typed.

The form of the UNITSTATUS call is

```
UNITSTATUS (UNITNUM, PAB, CONTROL);
```


- Where UNITNUM is an expression with an integer value that is the unit number of a particular input/output device, in this case of the console;

- Where PAB is a packed array of a type you name, called BYTE, which is meant to hold the keyboard data you want from the console driver;

- Where CONTROL is an expression with an integer value referencing individual bits to control the operation of the UNITSTATUS procedure.

The CONTROL parameter tells the procedure that you want a transfer of status information in the keyboard buffer from the console unit device (parameter UNITNUM) to the packed array (PAB), where it can be read.

When you place a UNITSTATUS call in your program to check the status of the OPEN-APPLE, SOLID-APPLE, or SHIFT key, or any combination of them, insert these parameter values in the statement:

1. For UNITNUM, use the integer "2", the unit number of the keyboard device.

2. For PAB, use a packed array of byte--in the example that follows, "NCHARSBUFD[Ø]", which is the first field of the record variable KEYSTAT.

3. For CONTROL, use the integer "1", which specifies that you want to get status information on the console input rather than to perform a control operation on the console input or output.

Note that the procedure UNITSTATUS does not have to be defined because it is built into the Pascal language.

ECHO is a program demonstrating how a variable of type KEYSTAT is used with the UNITSTATUS procedure to check for the use of one or more of these special keys along with a character key.

```
PROGRAM ECHO;

TYPE    BYTE = Ø..255;

VAR     KEY :CHAR;
        KEYSTAT :RECORD
          NCHARSBUFD :PACKED ARRAY [Ø..1] OF BYTE;
          SHIFT, SOLIDAPPLE, OPENAPPLE : BOOLEAN
        END;

BEGIN
  WITH KEYSTAT DO
    REPEAT
      UNITSTATUS (2, NCHARSBUFD[Ø], 1);
      IF NCHARSBUFD[Ø] > Ø THEN
          BEGIN
            READ (KEYBOARD, KEY);
            WRITE (KEY, ' = ', ORD(KEY), '    ');
            IF OPENAPPLE THEN WRITE ('OPEN APPLE    ');
            IF SOLIDAPPLE THEN WRITE ('SOLID APPLE    ');
            IF SHIFT THEN WRITE ('SHIFT    ');
            WRITELN
          END;
    UNTIL KEY = 'Q'
END.
```

This demonstration program displays the character typed, as well as the ordinal value of the character in the ASCII sequence of characters. In addition, if any of the three special keys has been pressed along with the character key, it displays the names of the keys pressed. (Normally, your own ECHO procedure would not include the WRITE statements, which here act as an illustration, but would substitute a statement using the READ and UNITSTATUS outputs in a way appropriate to your application.)

THREE SPECIAL IDENTIFICATION FLAGS

Pascal 1.2 includes three special flags you may use to identify

1. Whether the computer in use is an Apple II, or an Apple IIe with or without an 8Ø-Column Text Card or Extended 8Ø-Column Text Card;

2. Whether the computer in use is executing the Pascal 1.1 system or the Pascal 1.2 system;

3. Which version of the Pascal interpreter is executing on the computer in use--the 48K, the 64K, or the 128K--and whether certain other variations are operative.

FLAG TO CHECK THE COMPUTER TYPE

By identifying which machine it is running on, an application program for the Apple IIe can take advantage of the unique features of the Apple IIe but retain the capacity to run on the Apple II. Memory location -16591 ($BF31 hexadecimal) contains a flag you may use to determine from within a program whether the computer is an Apple II or an Apple IIe. If the flag is set with the high-order bit (bit 7) turned on, the machine is an Apple IIe. If bit 7 is turned off--the machine is an Apple II. (There is a possible third case--where bit 6 and bit 7 are both turned on--for a machine that is neither an Apple II or IIe but is Apple IIe compatible. This possibility is currently inactive but reserved should Apple Computer, Inc. later manufacture a computer that is not an Apple II or IIe but is Apple IIe compatible.)

If the machine is an Apple IIe, this same memory location will also specify whether the computer has an 8Ø-column card and whether it also has the auxiliary 64K of RAM memory available on the Apple Extended 8Ø-Column Text Card.

The flag bit settings listed in Table 4-4 are made whenever the Pascal system starts up.

If the computer	then the byte at memory location -16591 ($BF31 Hex) has these bit settings: Bit 7	Bit 6	Bit 1	Bit Ø
Is an Apple IIe	ON	OFF	OFF	OFF
And has an 8Ø-column card	ON	OFF	OFF	ON
With 64K auxiliary card RAM	ON	OFF	ON	ON
Is an Apple II	OFF	OFF	OFF	OFF

Table 4-4. Hardware Identification Bit Settings

There is no built-in Pascal routine to access this flag byte. You have to use a Pascal variant record or a assembly language routine.

CHECKID is a program, which you could implement as a procedure, using a variant record to determine whether the computer is an Apple IIe and whether it has an 8Ø-column card with or without the auxiliary 64K RAM.

```
PROGRAM CHECKID;

TYPE      BITARRAY = PACKED ARRAY [Ø..7] OF BOOLEAN;
          ADDR = RECORD
            CASE BOOLEAN OF
              FALSE:  (INT:INTEGER);
              TRUE:   (PTR:^BITARRAY);
            END;

VAR       A:ADDR;
          IIE, EIGHTYCOLS, AUX64KMEM:BOOLEAN;

BEGIN
  ...

  A.INT := -16591;                    {location $BF31 hex}
  IIE := (ADDR.PTR^[7]) AND (NOT ADDR.PTR^[6]);
  EIGHTYCOLS := ADDR.PTR^[Ø];
  AUX64KMEM := ADDR.PTR^[1];
  ...

END.
```

If the computer is an Apple IIe, variable IIE will have a value of True. If the computer is an Apple IIe and has an 8Ø-column card without the 64K RAM, variables IIE and EIGHTYCOLS will have a value of True. If the computer is an Apple IIe and has an 8Ø-column card with the 64K RAM, the variables IIE, EIGHTYCOLS, and AUX64KMEM will have a value of True. If the computer is an Apple II, the three variables will all have a value of False.

FLAG TO CHECK THE PASCAL SYSTEM VERSION

When Pascal 1.2 is started up on an Apple II or IIe, a flag is set at memory address -166Ø7 ($BF21 hex) to identify which Pascal version is the one being used.

- If Pascal 1.2 is operating, the value of the byte at that location is "3".
- If Pascal 1.1 is operating, the value of the byte at that location is "2".

FLAG TO CHECK THE INTERPRETER VERSION

To identify which Pascal interpreter is executing, another flag is set at startup time--at memory address -166Ø6 ($BF22 hex). This flag uses different bit settings to identify the variations being supported, as Table 4-5 shows.

Bit Number	Set to Value	Indicates
Ø	Ø	The Pascal development system is executing.
Ø	1	The Pascal run-time system is executing.
1	1	Floating-point operations are not supported.
2	1	Operations using sets are not supported.
5	1	The 48K Pascal interpreter is executing.
6	Ø	
5	Ø	The 64K Pascal interpreter is executing.
6	Ø	
5	Ø	The 128K Pascal interpreter is executing.
6	1	
5	1	This bit setting is reserved for future system use.
6	1	
7	Ø	All console output is directed to the text screen pages, an external terminal, or an 8Ø-column card.
7	1	All console output is directed to the high-resolution pages.

Table 4-5. Version Flags Set at Location -166Ø6 ($BF22 Hex)

TWO IMPORTANT POINTER LOCATIONS

Table 4-6 tells you where these two important Pascal pointers are located and describes them. Although you cannot change their values, because they are a fixed part of the Pascal system, you can use variant records to see the current value in either location.

Location on Zero-Page	Pointer	Its Purpose
$5A	NP	Points to top of Pascal data heap.
$5C	KP	Points to top of Pascal program stack.

Table 4-6. Two Pascal Pointers

NEW VALUES FOR THE UP-ARROW AND DOWN-ARROW KEYS

The Apple IIe UP-ARROW and DOWN-ARROW keys are hardwired to the ASCII values for CONTROL-K and CONTROL-J, respectively. The definitions of "up cursor" and "down cursor" have been changed in the Pascal 1.2 SYSTEM.MISCINFO file to the same values.

To make the 40-column Apple II behave as under Pascal 1.1--that is, to make CONTROL-O and CONTROL-L move the cursor up and down, respectively--Pascal 1.2 causes the keystrokes CONTROL-O and CONTROL-L to produce the same ASCII values as the keystrokes CONTROL-K and CONTROL-J, respectively.

As a result of this alteration of key values, the ASCII values for CONTROL-O and CONTROL-L can no longer be produced by typing on the keyboard of the Apple II.

However, Pascal 1.2 users of an 80-column card on an Apple II may need to use the old ASCII values produced by CONTROL-O and CONTROL-L. If so, they should move to their startup disk the special II80.MISCINFO file found on the APPLE3: disk of their Pascal 1.2 system. (For instructions on how to do this, see the section "Special MISCINFO Files and How to Use Them" in Chapter 2.)

If you try to use an application program (designed to run on an Apple II in 40-column mode) that makes hard-coded checks for the old up-cursor and down-cursor keyboard values, the program will not work properly under Pascal 1.2. Such a program will have to be changed to obtain the new key values (those for CONTROL-K and CONTROL-J) from the Pascal 1.2 SYSTEM.MISCINFO file at load time. The next section shows how to do this.

READING THE UP-CURSOR AND DOWN-CURSOR VALUES

To read the up-cursor and down-cursor values from the SYSTEM.MISCINFO file, you need only know that the

- up-cursor value (KEY TO MOVE CURSOR UP) is at byte 78, and the
- down-cursor value (KEY TO MOVE CURSOR DOWN) is at byte 79

of the SYSTEM.MISCINFO file.

The following program fragment illustrates the use of this information:

```
VAR UpKey       : CHAR;
    DownKey     : CHAR;
    CH          : CHAR;

  PROCEDURE SetKeys;
    VAR buf     : PACKED ARRAY [0..511] OF CHAR;
        f       : FILE;
  BEGIN
      RESET (f, '*SYSTEM.MISCINFO');
      IF BLOCKREAD(f, buf, 1) <> 1 THEN BEGIN
          writeln ('Unable to read 1 block of *SYSTEM.MISCINFO');
          exit (PROGRAM)
      END {if};
      UpKey   := buf[78];
      DownKey := buf[79]
  END {SetKeys};

...

BEGIN {main program}
    SetKeys;
    ...

    READ (KEYBOARD, CH);
    IF   CH=UpKey THEN <whatever the program does for up-arrows>
    ELSE IF CH=DownKey THEN <ditto for down-arrows>
    ELSE etc
    ...
END {main program}.
```

Note that this routine is suitable for the Apple II and Apple IIe, with or without an 80-column card, but does not work for any external terminal that generates multiple-character sequences such as ESCAPE-U and ESCAPE-D for arrow keys.

CHANGES TO THE SEEK AND PUT PROCEDURES

The Pascal system's SEEK and PUT procedures now work differently from the way they worked under Pascal 1.1. Here are the changes:

1. In Pascal 1.1, if a program directs SEEK to find a record that is past the last byte used in the last block of a disk file, SEEK positions the file pointer at the last byte used in the last block.

   In Pascal 1.2, if a program directs SEEK to find a record that is past the last byte used in the last block of a disk file, SEEK first tries to expand the file. If the file cannot be expanded--that is, if there is another file starting right after it--the system returns an IORESULT of 8 ("no room"). If the file can be expanded, SEEK does so. Then SEEK checks to see whether the record it originally was directed to find is within the new bounds of the file. If it is not, the system returns an IORESULT of 8. If it is within the new bounds, SEEK positions the file pointer at the record.

2. In Pascal 1.1, if a program directs SEEK to a position anywhere between the current EOF and the EOF at the time the file was last RESET, SEEK does not refill its file buffer with the correct block. Consequently, if the position being sought is not in the same block as the pre-SEEK position, SEEK will not position the file pointer correctly.

   In Pascal 1.2, SEEK refills its file buffer with the correct block and positions the file pointer correctly.

3. In Pascal 1.1, if a program attempts to extend a disk file dynamically by seeking to a record number that is after the last record in a file immediately followed by another file, and then doing a PUT, the first block of the following file will be damaged. The cause of the damage is this: SEEK moves the file pointer to the last byte used in the last block of the file. (See 1, above). Then PUT discovers that it is positioned at the end of the file, so it tries to expand the file in order to put the new record in. If the file can be expanded, PUT does so. Unfortunately, PUT then adds the record without first checking to see whether the expanded file was expanded enough to hold the record. If it wasn't, the first block of the following file is overwritten.

   In Pascal 1.2, if PUT has been able to expand the file, it checks to make sure that the record will actually fit in the remaining space in the file without overlapping the file that follows. If there is not enough space for the record, the system returns an IORESULT of 8 ("no room").

4. In Pascal 1.1, the ability to seek beyond EOF is inconsistent, because SEEK never attempts to expand a file, although PUT does. Consequently, sometimes SEEK will succeed in positioning the pointer after the last record in the file, but other times it will not.

   In Pascal 1.2, the ability to seek beyond EOF is consistent because both SEEK and PUT attempt to expand the file if necessary. (See 1, above.)

You will find a complete list of bug fixes in Appendix A in this manual.

TWO FEATURES NO LONGER OPERATIVE

To streamline the memory requirement of the Pascal 1.2 operating system, we removed two previously available features.

First, the operating system ignores the "has slow terminal" flag in SYSTEM.MISCINFO as a feature that is no longer useful.

Second, the system assumes that every console device presently in use has a "backspace" value (usually ASCII 8).

APPENDIX A

BUG FIXES IN PASCAL 1.2

77 COMPILER BUGS
78 ASSEMBLER BUGS
79 LINKER BUGS
79 LIBRARY.CODE BUG
79 LIBMAP.CODE BUG
79 SEEK/PUT BUGS
80 INPUT/OUTPUT BUGS
80 TURTLEGRAPHICS BUGS
81 MISCELLANEOUS EXECUTION-TIME BUGS

APPENDIX A
BUG FIXES IN PASCAL 1.2

The following bugs in Pascal 1.1 have been corrected in Pascal 1.2:

COMPILER BUGS

1. A regular unit using (*$R segname*) or (*$R unitname*) was not linked properly. Now it is.

2. If the Compiler Resident option ($R) was done on an intrinsic unit which has a data segment, the code segment was loaded before the data segment. Consequently, assembly language routines containing addresses of items in the data segment did not have those addresses correctly relocated. Now the data segment is loaded first, and addresses are correctly relocated.

3. EXIT(procedurename) did not work if the procedure was in a regular unit and had a procedure number greater than 127. Now it works correctly.

4. Initialization sections of nested units were (incorrectly) executed in the reverse order. Now they are executed in the correct order.

5. The Compiler did not issue an error message if (*$R segname*) or (*$R unitname*) referred to a non-existent segment or unit. Now it does.

6. The Compiler did not issue an error message if an intrinsic unit had the same segment number for code and data segments. Now it gives an error message.

7. The Compiler did not check for an empty data segment in an intrinsic unit. Now it does.

8. The Compiler issued an error #35Ø message ("No data segment allocated") at the end of the initialization section, instead of at the beginning of the implementation, if an intrinsic unit needed a data segment but none was declared in the UNIT statement. Now it gives this error message at the beginning of the implementation.

9. If a procedure was declared FORWARD but never defined, the Compiler error message did not stop the compile and was easily overlooked as the display scrolled. The compile looked successful but did not create a code file. Now the compile is stopped, and the error message is clear.

1Ø. The Compiler sometimes released symbol table space from the heap too soon. Now it does it at the proper time.

11. Negative long integer constants and variables were not processed correctly. Now they are.

12. The Compiler did not test for a constant string longer than 8Ø characters. Now it does.

13. When a listing was turned off, the Compiler continued to emit a form feed for every (*$P*) encountered. Now it does not.

14. The Compiler could get random errors on identifiers that began with the letters H, J, K, Q, X, Y, or Z. Now it does not.

15. The Compiler would ingore all Compiler options that followed $S++ if they were in a single statement. Now it does not.

ASSEMBLER BUGS

1. The Assembler did not process .ALIGN properly. Now it does.

2. If there were no symbols in the program, the Assembler would print garbage for the symbol table. Now it does not.

3. A fixup to a word that crossed buffer boundaries would destroy the byte following the end of the buffer. Now it does not.

4. The Assembler did not test for nested macros, which are illegal. Now it does test and gives an appropriate error message if necessary.

5. An .ENDM without a corresponding .MACRO was not printed, and caused the Assembler to give inappropriate "Undefined label" errors.

6. If an assembly contained more than 1Ø procedures, the Assembler could overwrite SYSTEM.PASCAL data. Now it does not.

7. The Assembler did not correctly relocate .INTERP references. Now it does.

8. If there was garbage after the filename in an .INCLUDE statement, the Assembler reported the error but continued to the next line without including the file. Now the Assembler includes the file after reporting the error.

LINKER BUGS

1. The Linker sometimes failed to resolve DEFs and REFs properly to see that the labels of two assembly routines match. Now it resolves them correctly to see that the labels match.

2. The system did not allow regular units to use segments numbered in the range 16..31 . Now it does.

LIBRARY.CODE BUG

1. Once the library file being built exceeded 2ØØ blocks, the interface text sections of units were no longer copied into the file. Now it copies them into the file.

LIBMAP.CODE BUG

1. LIBMAP.CODE did not list in the output file the interface text of units that started after block 2ØØ of the library. Now they are listed.

SEEK/PUT BUGS

1. An attempt to extend a disk file dynamically by doing a SEEK to a record number falling after the final block in the file, when the space immediately after the file is occupied by another file, and then doing a PUT, resulted in writing over the first block of the following file. Now the PUT following the SEEK will get an IORESULT of 8 ("no room") if there is not enough space for the record. (See Chapter 4 of this manual for details.)

2. If there was room in the last block of a file to contain another record, a SEEK to a record number that should have resulted in expansion of the file resulted, instead, in the record being written following the former last record. Now the system first tries to expand the file: if there is not enough space for the record, it gives an IORESULT of 8 ("no room"). (See Chapter 4 of this manual for details.)

3. SEEK went to the wrong position whenever the program tried to seek to a position anywhere between the current EOF and the EOF at the time of the last RESET. Now SEEK refills its file buffer with the correct block and positions the pointer correctly. (See Chapter 4 of this manual for details.)

4. A SEEK to a large record number in a file with a large record size would take an inordinate amount of time. For example, SEEK(f,12ØØØ), where "f" contains 25Ø-byte records, took about 15 seconds. Now such a SEEK takes a much shorter time.

INPUT/OUTPUT BUGS

1. If a driver was attached to a unit number in the range 4..12, UNITREADS and UNITWRITES would work correctly, but file input or output using the volume name might not work correctly. Now file input or output using the volume name works correctly.

2. The flexible disk routines disabled interrupts and left them disabled. Now program interrupts are re-enabled after the disk routines finish.

3. A WRITE of a null string did not set IORESULT properly if IORESULT was non-zero before the WRITE. Now it does.

4. RESET or REWRITE of a file on a DOS-formatted volume did not return an IORESULT error, as it should have. Now they return an IORESULT of 1Ø ("File not found").

TURTLEGRAPHICS BUGS

1. The Parameter YSKIP in the procedure DRAWBLOCK did not work. Now it does.

2. The DRAWBLOCK procedure did not draw the block correctly if part of the block extended past the edge of the viewport. Now it does.

MISCELLANEOUS EXECUTION-TIME BUGS

1. If the (*$NR*) option referred to a unit that had an assembly language procedure in its INTERFACE, and that procedure was called from outside the unit, a fresh copy of the code segment was loaded onto the stack as it should be. But when the procedure terminated, the storage occupied by that segment was not released. Repeated calls would then cause memory to fill with copies of the unit's code. Now the storage is released properly.

2. Repeated CONTROL-A's during compiles (to look at the second forty columns of the screen) could crash the system, and repeated CONTROL-A's during disk input/output could cause bad disk writes. Now such use of CONTROL-A does not crash the system or cause bad disk writes.

3. If the "ignore external terminal" flag was set with RTSETMODE, the card in slot 3 was initialized even though it was not used afterward. Now the card in slot 3 is not initialized.

4. Sometimes when a program wrote a real number, using a wide field, a random character was written in the rightmost position of the field. Now the number is written correctly.

5. If an error occurred on a long-integer computation, the interpreter entered a loop instead of giving an execution error. Now it gives an execution error.

6. There was no check for stack overflow when intrinsic-unit data segments were loaded on the stack. Now there is.

APPENDIX B

THE FILES ON THE PASCAL 1.2 DISKS

The following list gives the files on each of the flexible disks for Pascal 1.2. The order of files on any disk is unimportant. Files that are not included in version 1.1 of Pascal are marked with a single asterisk (*). If a file is on a disk in a place different from where it was in Pascal 1.1, it is marked with a double asterisk (**). For detailed information about the various files and their use, see Appendix D of the Apple Pascal Operating System Reference Manual.

APPLEØ: disk

SYSTEM.PASCAL
SYSTEM.COMPILER
SYSTEM.EDITOR
SYSTEM.FILER
SYSTEM.LIBRARY
SYSTEM.CHARSET
SYSTEM.MISCINFO
SYSTEM.SYNTAX

APPLE1: disk

SYSTEM.APPLE
SYSTEM.PASCAL
SYSTEM.EDITOR
SYSTEM.FILER
SYSTEM.LIBRARY
SYSTEM.CHARSET
SYSTEM.MISCINFO
SYSTEM.SYNTAX

APPLE2: disk

SYSTEM.COMPILER
SYSTEM.LINKER
SYSTEM.ASSMBLER
LIBRARY.CODE**
LIBMAP.CODE**

```
65ØØ.OPCODES
65ØØ.ERRORS

APPLE3: disk

SYSTEM.APPLE
BINDER.CODE
LINEFEED.TEXT
LINEFEED.CODE
FORMATTER.CODE
FORMATTER.DATA
BALANCED.TEXT
CROSSREF.TEXT
DISKIO.TEXT
GRAFCHARS.TEXT
GRAFDEMO.TEXT
HAZELGOTO.TEXT
HAZEL.MISCINFO
HILBERT.TEXT
II4Ø.MISCINFO*
II8Ø.MISCINFO*
IIE4Ø.MISCINFO*
SETUP.CODE
SPIRODEMO.TEXT
TREE.TEXT
128K.APPLE*
128K.PASCAL*
```


APPENDIX C

ERROR MESSAGES

87 COMPILER ERROR MESSAGES
90 NEW ASSEMBLER ERROR MESSAGE

APPENDIX C

The Compiler error messages have been revised for Pascal 1.2. Following is an updated list of all the Compiler error messages, as well as the one new Assembler error message, in Pascal 1.2:

COMPILER ERROR MESSAGES

```
 1: Error in simple type
 2: Identifier expected
 3: 'PROGRAM' expected
 4: ')' expected
 5: ':' expected
 6: Illegal symbol (maybe missing or extra ';' on line above)
 7: Error in parameter list
 8: 'OF' expected
 9: '(' expected
1Ø: Error in type
11: '[' expected
12: ']' expected
13: 'END' expected
14: ';' expected (possibly on line above)
15: Integer expected
16: '=' expected
17: 'BEGIN' expected
18: Error in declaration part
19: Error in field-list
2Ø: ',' expected
21: '.' expected
22: 'Interface' expected
23: 'Implementation' expected
24: 'CODE' expected
5Ø: Error in constant
51: ':=' expected
52: 'THEN' expected
53: 'UNTIL' expected
54: 'DO' expected
55: 'TO' or 'DOWNTO' expected in FOR statement
58: Error in factor (bad expression)
```

```
 59: Error in variable
101: Identifier declared twice
102: Low bound exceeds high bound
103: Identifier is not of the appropriate class
104: Undeclared identifier
105: Sign not allowed
106: Number expected
107: Incompatible subrange types
108: File not allowed here
109: Type must not be real
110: Tagfield type must be scalar or subrange
111: Incompatible with tagfield part
113: Index type must be a scalar or a subrange
114: Base type must not be real
115: Base type must be a scalar or a subrange
117: Unsatisfied forward reference
119: Re-specified params not OK for a forward declared procedure
120: Function result type must be scalar, subrange or pointer
121: File value parameter not allowed
122: Result type of forward declared function cannot be re-specified
123: Missing result type in function declaration
125: Error in type of standard procedure parameter
126: Number of parameters does not agree with declaration
128: Result type does not agree with declaration
129: Type conflict of operands
130: Expression is not of set type
131: Only tests on equality are allowed
132: Strict inclusion not allowed
133: File comparison not allowed
134: Illegal type of operand(s)
135: Type of operand must be boolean
136: Set element type must be scalar or subrange
137: Set element types must be compatible
138: Type of variable is not array
139: Index type is not compatible with the declaration
140: Type of variable is not record
141: Type of variable must be file or pointer
142: Illegal actual parameter
143: Illegal type of loop control variable
144: Illegal type of expression
145: Type conflict
146: Assignment of files not allowed
147: Label type incompatible with selecting expression
148: Subrange bounds must be scalar
149: Index type must not be integer
150: Assignment to standard function is not allowed
152: No such field in this record
154: Actual parameter must be a variable
155: Control variable cannot be formal or non-local
156: Multidefined case label
158: No such variant in this record
159: Real or string tagfields not allowed
160: Previous declaration was not forward
161: Forward declared twice
```



```
162: Parameter size must be constant
165: Multidefined label
166: Multideclared label
167: Undeclared label
168: Undefined label
169: Base type of set too large
175: Actual parameter max string length < formal max length
182: Nested units not allowed
183: External declaration not allowed at this nesting level
184: External declaration not allowed in interface section
185: Segment declaration not allowed in unit
186: Labels not allowed in interface section
187: Attempt to open library unsuccessful
188: Unit not declared in previous uses declaration
189: 'Uses' not allowed at this nesting level
190: Unit not in library
191: No private files in unit
192: 'Uses' must be in interface section
194: Comment must appear at top of program
195: Unit not importable (interface text not available)
201: Error in real number--digit expected
202: String constant must not exceed source line
203: Integer constant exceeds range
250: Too many scopes of nested identifiers
251: Too many nested procedures or functions
253: Procedure too long
254: Procedure too complex
273: No such unit or segment
277: String too long
301: No case provided for this value
350: No data segment allocated
352: No code segment allocated
353: Non-intrinsic unit called from intrinsic unit
354: Too many segments for segment dictionary
355: Data segment empty
399: Implementation restriction
400: Illegal character in text
401: Unexpected end of input
402: Error in write to code file, maybe not enough room on disk
403: Error while opening or reading include file
404: Bad open, read, or write to Linker file SYSTEM.INFO
405: Error while reading library
406: Include file not legal in interface nor while including
408: (*$S+*) needed to compile units
409: General Compiler error
```

NEW ASSEMBLER ERROR MESSAGE

65: Too many .PROCS and/or .FUNCS

APPENDIX D
ACTIVATING THE SHIFT-KEY MOD

If you want the SHIFT-key mod installed on your Apple II or IIe, see your dealer. The only reason for having it installed on an Apple IIe is so that a program can test to see if the user has pressed the SHIFT key alone or in conjunction with another key in response to a program prompt. Having it installed on an Apple II allows a program to test for the SHIFT key, allows you to shift between uppercase and lowercase characters using the SHIFT key in the normal fashion, and causes certain keyboard character translations, where, for example, typing SHIFT-P produces an uppercase P instead of @ .

Having the SHIFT-key mod installed on an Apple II does not mean that the modification automatically works. Nor is it activated automatically at startup time. You must activate the modification in order to take advantage of some of its functions. You may also deactivate the modification when desirable. The modification does not have to be activated on an Apple IIe.

- You activate the modification by pressing either CONTROL-E or CONTROL-W after starting up an application or the Pascal system.

- You deactivate the modification by pressing CONTROL-T.

Activating the modification on an Apple II allows the SHIFT key to be used to obtain uppercase and lowercase characters in the conventional shift-key manner. But two functions do not require modification activation: using and testing the SHIFT key as a control, and obtaining certain new character translations. Table D-1 lists the keyboard character translations on an Apple II that are a direct effect of the modification and that remain in effect before and after it is activated.

Typing or Pressing These Keys	Obtains This Character
^	N
@	P
]	M
CTRL-^	^
CTRL-@	@
CTRL-]	]

Table D-1. SHIFT-Key Mod Character Translations

Table D-2 summarizes what results from the modification when it is not activated and what results when it is activated.

Mod Installed but Not Active	Mod Installed and Active (Apple II Only)
The character translations resulting from the mod become operative--Apple II only.	The character translations remain operative.
On an Apple II and an Apple IIe, the SHIFT key can be tested using the UNITSTATUS procedure.	The SHIFT key can be tested using the UNITSTATUS procedure. Lowercase and uppercase characters can be obtained using the SHIFT key.

Table D-2. Effects of an Activated and Inactivated SHIFT-Key Mod

INDEX

% (percent sign) prefix 15
  in Library Name File 49
128K system 29-51
  error messages 32
  features 29
  memory organization 33
  setting up 30
  startup disk 30
40-column screen
  on Apple II 22-23, 24
  on Apple IIe 22-23, 25
80-column card, with Apple II 70
80-column screen
  on Apple II 22-23, 25
  on Apple IIe 22-23
{$S=} option 38

A

accessing files during program execution 15
Apple II, defined 4
Apple II Plus 4
Apple IIe 4
Apple Pascal Language Reference Manual 4
Apple Pascal Operating System Reference Manual 4
APPLE0: disk 4, 83
APPLE1: disk 4, 83
APPLE2: disk 4, 83
APPLE3: disk 4, 84
APPLESTUFF unit 42, 57
arrow keys 70
Assembler
  bug fixes 78
  error message 90
assembly code 34, 35, 36
auxiliary memory 29, 37, 38-39
  managing 38-39

B

backing up disks 6
backspace value 73
block volume units 19
BLOCKREAD procedure 55
BLOCKWRITE procedure 55
boot, two-stage 14
boot disk, request for 14
break in execution 21
bug fixes 77-81
  Assembler 78
  Compiler 77-78
  execution-time 81
  Input/Output 80
  LIBMAP.CODE 79
  LIBRARY.CODE 79
  Linker 79
  SEEK/PUT 79-80
  Turtlegraphics 80
button 0 60
button 1 60

C

CAPS LOCK key 20
chaining 17
CHAINSTUFF unit 8, 55
characters
  reading and writing 57
  screen-control 59
code file, executable 41
code overflow 32
.CODE suffix 43
CODELOW variable 38, 39
CODEP variable 38, 39
"Codespace overflow" message 32
Compiler 38, 40
  bug fixes 77-78
  error messages 87-89
computer type flag 67
conserving disk space 44
control characters not echoed to the screen 19
CONTROL-@ 21
CONTROL-] 20
CONTROL-E 20, 91
CONTROL-J 70
CONTROL-K 21, 70
CONTROL-L 21, 70
CONTROL-O 21, 70
CONTROL-R 20
CONTROL-T 21, 91
CONTROL-W 20, 91
CONTROL-OPEN-APPLE-RESET 9
CONTROL-RESET 9
CONTROL-SHIFT-2 21
copying disks 6
cursor motion keys 8, 20, 22

D

data files, specifying using % 15
data space in memory 34, 35, 36
DEL character 21
DELETE key 21
devices, remote 57
disk space, conserving 44
disk-formatting program 13
DOWN-ARROW key 70
down-cursor key 8
down-cursor value 71

E

Editor, Pascal 38
error messages 87-90
executable code file 41
execution-time bug fixes 81
Extended 80-Column Text Card 29, 32, 33, 37, 38

F

file pathname 40, 41
filename 15
Filer, Pascal 38
files
  on Pascal 1.2 disks 83
  library 40
flags
  computer type 67
  has slow terminal 73
  identification 66-69
  ignore external terminal 59
  Pascal interpreter 68
  Pascal version 68
flexible disk 15
floppy disk 15
foreign keyboards 21
Formatter 13
  error messages 14

G

game paddle mod 61
GET procedure 18, 55

H

Halt command 9
hand-control buttons 19
"has lower case" control variable 22
"has slow terminal" flag 73
heap pointer 70
heap, Pascal data 34, 35, 36
hexadecimal number system 33
high-bit test for OPEN-APPLE key 61-63

I

I/O 34, 35, 36, 37
identification flags 66-69
ignore external terminal flag 59
II40.MISCINFO file 22
II80.MISCINFO file 22
IIE40.MISCINFO file 22
Initialize command 9
Input/Output 34, 35, 36, 37
Input/Output bug fixes 80
interpreter, Pascal 34, 35, 36
intrinsic units 42
inverse video
  turning off 59
  turning on 59

J

K

keyboard, foreign 21

L

Language Card 33, 34
.LIB suffix 43
LIBMAP.CODE 40
LIBMAP.CODE bug fixes 79
library file(s) 40, 42
  more than one with a program 47
  options 40, 45
  searching in 50
  sharing 44, 46
Library File, Program 43
Library Name File 43
  % prefix in 49
  making 45
  using 46-49
LIBRARY utility code 42
LIBRARY.CODE 40
  bug fixes 79
Linker, Pascal 40
  bug fixes 79
LONGINTIO unit 8
lowercase characters 20
  "has lower case" variable 22

M

main memory 38, 34, 35
memory, getting more 17
memory maps 33-37
memory organization, features 38
MISCINFO files, special 22
mixing Pascal 1.1 and 1.2 7
modification, hardware 60-61, 91

N

Name File, Library 43
  % prefix in 49
  making 45
  using 46-49

O

OPEN-APPLE key 19, 60
  high-bit test for 61-63
operating system 38, 34, 35, 36, 37
overflow, code 32

P

P-code 38, 34, 35, 37
Pascal 1.1 4
  running programs under 1.2 8
  upgrading SYSTEM.LIBRARY 8
Pascal 1.2
  disks 6, 7
  operating system 7
  reasons to use 4
Pascal 128K system 29-51
  error messages 32
  features 29
  memory organization 33
  setting up 30
  startup disk 30
Pascal 64K system memory map 33, 34-35
Pascal data heap 34, 35, 36
Pascal interpreter 34, 35, 36
Pascal interpreter flag 68
Pascal operating system 38, 34, 35, 36, 37
Pascal prefix 48
Pascal program stack 34, 35, 36
Pascal startup screen 14
Pascal version flag 68
PASCALIO unit 8
pathname 40, 41
percent prefix (%) 15-17
  accessing files 16
  chaining 17
  in Library Name File 49
pointer
  heap 70
  stack 70
pointer locations 69
Program Library File 43
program segments, too many 19
program units
  intrinsic 42
  regular 42
PUT procedure 18, 55
  changes 72

Q

R

READ procedure 55
reading up- and down-cursor values 71
regular units 42
remote devices 57
REMSTATUS function 57
reserved segments 40
rigid disk 15
RSTATTYPE value 57
RTSETMODE program 59

S

{$S=} option 38
saving disk space 44
screen-control characters 59
searching in library files 50
SEEK procedure changes 72
SEEK/PUT bug fixes 79-80
segments
  additional with 128K 40
  reserved 40
SETCHAIN procedure call 56
sharing library files 46
SHIFT key 19, 60
SHIFT-key mod 61, 91-92
SOLID-APPLE key 19, 60
special MISCINFO files 22
"Stack overflow" message 32
stack pointer 70
stack, Pascal program 34, 35, 36
startup
  one-drive system 9
  two-drive system 9
  two-stage 14
startup disk
  128K system 30
  request for 14
startup screen 14
SWAPGPON procedure 55
SWAPOFF procedure 55
SWAPON procedure 55
swapping during compilation 38
swapping option(s)
  program level 55-57
  system level 17-18
system size, finding 14
SYSTEM.APPLE file 29
SYSTEM.LIBRARY file 42, 57
  upgrading 8
SYSTEM.MISCINFO file 22
SYSTEM.PASCAL file 7, 29

T

tests
  high-bit for OPEN-APPLE 61-63
  UNITSTATUS 64-65
TURTLEGRAPHICS unit 8
Turtlegraphics bug fixes 80

U

unit(s)
  block volume 19
  intrinsic 42
  regular 42
unit number 40, 41
UNITREAD procedure 55
UNITSTATUS test 64-65
UNITWRITE procedure 55
UP-ARROW key 70
up-cursor key 8
up-cursor value 71
uppercase characters 20
USES APPLESTUFF declaration 58
USES CHAINSTUFF declaration 56

V

volume name 15, 40, 41

W

WRITE procedure 55

X, Y

Z

zero-page pointer 39

UCSD PASCAL SYSTEM USER'S SOCIETY (USUS)

GET MORE FROM YOUR PASCAL SYSTEM
JOIN USUS TODAY

USUS is the USER'S GROUP for the most widely used, machine-independent software system.

If you use UCSD Pascal*, Apple Pascal**, or the UCSD p-System*, USUS will link you with a community of users who share your interests.

USUS was formed to give users an opportunity to promote and influence the development of UCSD Pascal and the UCSD p-System, and to help them learn more about their systems. USUS is non-profit, and is not affiliated with any vendor.

Members get access to the latest UCSD Pascal information, and to extensive Pascal expertise. In USUS, you have formal and informal opportunities to communicate with and learn from other users via the following membership benefits::

NATIONAL MEETINGS twice a year let you learn from experts and try out the newest products. Meetings feature hardware and software demonstrations, tutorials, technical presentations and information, reduced-cost software library access, SIG meetings, and a chance to ask questions of major vendors.

USUS NEWS AND REPORT brings you news and information about your operating system four times a year. It contains technical articles and updates, library catalog listings, SIG reports, a software vendor directory, and organizational news.

ELECTRONIC MAIL puts USUS subscribers in touch with a nationwide network of users. CompuServe MUSUS SIG provides a database and bulletin board communications. GTE Telemail accomodates one-to-one messages.

SOFTWARE EXCHANGE LIBRARY offers an extensive collection of tools, games, applications and aids, in UCSD Pascal source code at nominal prices.

SPECIAL INTEREST GROUPS zero in on specific problems, and represent member interests with manufacturers

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

** Apple Pascal is a trademark of Apple Computer Inc.

Disclaimer: The statement above is made by USUS. Its appearance here is for your information, and does not constitute an endorsement or warranty by Apple Computer.

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 of North America, air mail service is available for a surcharge. It is as follows: $5 annually for those in the Caribbean, Central America, Colombia, and Venezuela; $10 annually to those in South America, Turkey, and North Africa; and $15 for all others. Check or money order should be drawn on a U.S. bank or U.S. Post Office.

Name/Title _____

Affiliation _____

Address _____

_____

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: 6502/Apple _____ Other _____

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

_____ Advanced System Editor SIG
_____ Apple SIG
_____ Application Developer's SIG
_____ Communications SIG
_____ File Access SIG
_____ Graphics SIG
_____ Meetings Committee
_____ Modula-2 SIG
_____ Publications Committee
_____ Software Exchange Library
_____ Technical Issues Committee
_____ 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. Post Office to

Secretary, USUS
P.O. Box 1148
La Jolla, CA, 92038
USA