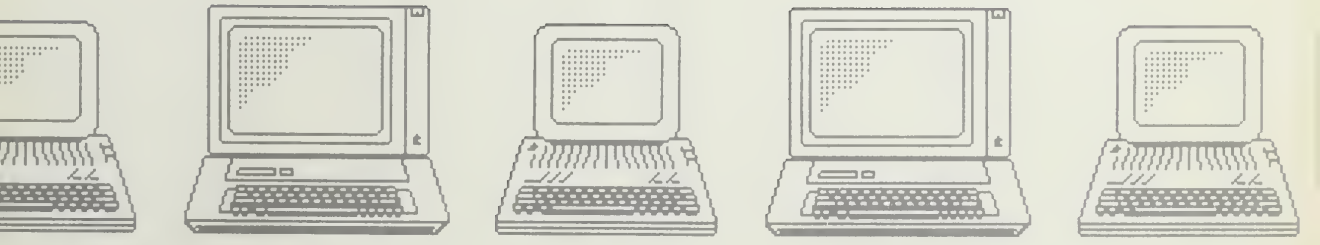
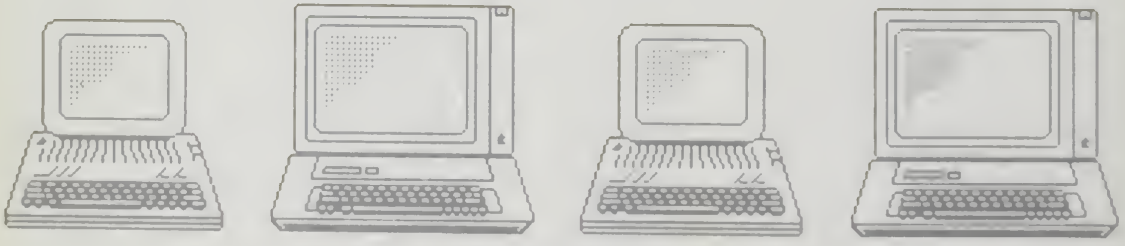


Sale of this material benefits the Library.
Boston Public Library.
This item is the property of the
BOSTON
PUBLIC
LIBRARY







BASIC Programming With ProDOS

Digitized by the Internet Archive
in 2012

http://archive.org/details/basicprogramming00read



BASIC Programming With ProDOS



Addison-Wesley Publishing Company, Inc.
Reading, Massachusetts Menlo Park, California Don Mills, Ontario
Wokingham, England Amsterdam Bonn Sydney Singapore Tokyo
Madrid Bogotá Santiago San Juan

Copyright © 1982, 1983, 1987 by Apple Computer, Inc.

This manual and the accompanying software (computer programs) are copyrighted by Apple or by Apple's suppliers, with all rights reserved. Under the copyright laws, this manual and software may not be copied, in whole or in 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 lent 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.

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

Printed in the United States of America. Simultaneously published in the United States and Canada.

ISBN 0-201-17755-2
First printing, March 1987
ABCDEFGHIJ-DO-8987

Warranty Information

ALL IMPLIED WARRANTIES ON THIS MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

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. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.



BASIC Programming With ProDOS

Contents

Figures, Tables, and Programs xx

Preface xxv
The BASIC Language and Disk Operating Systems xxv
The Parts of ProDOS xxv
Requirements for Using ProDOS xxvi
Important Note for Apple IIc Owners xxvii
This Manual's Organization xxviii
Watch for These xxix

CHAPTER 1 Introduction 1
Before You Start 2
Make a Spare Copy of *Examples* Disk 3
Starting ProDOS BASIC 3
The Startup Process 6
Other Ways to Start ProDOS BASIC 6
The HELP Command 7
Loading HELP 7
Using HELP 7
Using HELP Plus a Command Name 9
For Example 9
Installing HELP on Other Disks 11
Deleting HELP 11
ProDOS and DOS 11

CHAPTER 2

Files and Commands 13
About This Chapter 14
Disks 14
Files 14
Directory Files 15
Volume Directory Files 15
The /RAM Directory 16
For Example 16
Filenames 18
Some Legal Filenames 18
Some Illegal Filenames 18
Pathnames 19
The Prefix and Partial Pathnames 20
For Example 21
The General Form of ProDOS Commands 22
Options 22
Hexadecimal Notation 23
The Pathname Option —[pn] 23
For Example —[pn] 24
The Slot Number Option —[,S#] 24
The Drive Number Option —[,D#] 24
For Example —[,S#] [,D#] 25
In Summary 26

CHAPTER 3

Using Files 27
About This Chapter 28
This Chapter's Commands 28

The CAT and CATALOG Commands 30
The Options 34
For Example 35
What It All Means 36
The Directory's Name 37
NAME (Filenames) 37
TYPE (File Types) 37
BLOCKS (File Sizes) 38
MODIFIED and CREATED (File Dates) 38
ENDFILE (Maximum File Sizes) 38
SUBTYPE (File Properties) 38
The Bottom Line 39
The PREFIX Command 39
The Options 40
For Example 40
When You Use PREFIX in a Program 41
The CREATE Command 41
Directory File Size 42
The Options 42
For Example 43
The RENAME Command 44
The Options 44
For Example 45
The DELETE Command 45
The Options 46
For Example 46
The LOCK Command 46
The Options 47
For Example 47

The UNLOCK Command 48
The Options 48
For Example 48
The VERIFY Command 49
The Options 49

CHAPTER 4 BASIC Programs in Files 51
About This Chapter 52
BASIC Program Files 52
This Chapter's Commands 53
The —(DASH) Command 54
The Options 55
For Example 55
The RUN Command 56
The Options 57
For Example 57
The LOAD Command 58
The Options 59
For Example 59
The SAVE Command 60
The Options 60
For Example 61

CHAPTER 5 Programming With ProDOS 63
About This Chapter 64
This Chapter's Commands 64

What Is a Startup Disk? 68
The Anatomy of a Startup Disk 68
The Startup Process 69
Using ProDOS From Within a Program 71
For Example 72
Debugging Your Programs 72
Things to Watch Out For 72
You Can't Copy Control Characters 72
Some ProDOS Commands Work Only in Programs 73
Intercepting Messages to the Display Screen 73
Reading the Prefix 73
Handling Errors in a Program 74
Turning Off ONERR GOTO 76
Problems With ONERR GOTO 76
I/O From BASIC Programs 77
The CHAIN Command 78
The Options 78
For Example 79
The STORE Command 80
The Options 81
The RESTORE Command 81
The Options 81
For Example 82
The PR# Command 85
Starting Using PR# 86
The Options 86
For Example 86
The IN# Command 87
The Options 88

The FRE Command 88
For Example 88
The BYE Command 89
For Example 89

CHAPTER 6 Text in Files 91
About This Chapter 92
This Chapter's Commands 92
Sequential-Access Text Files: An Introduction 93
Random-Access Text Files: An Introduction 95
Sequential- and Random-Access Text Files 98
Position-in-the-File Pointer 99
Sequential Text Files 100
The Field 100
Storing Characters in Fields 100
A Simple Sequential Text File 101
Writing to a File Using PRINT 103
Reading Characters From a File 104
One Element Per Field 105
Multiple Elements Per Field 107
GET Characters From a File 108
Entering and Reading Text 110
A Program for Entering Text 111
A Program for Retrieving Text 113
The OPEN Command 115
The Options 116
For Example 116
Delete Before Opening 116

The CLOSE Command 117
The Options 117
The WRITE Command 118
The Options 118
The READ Command 119
The Options 119
The APPEND Command 120
The Options 120
For Example 120
The FLUSH Command 121
The Options 121
For Example 121
The POSITION Command 122
The Options 122

CHAPTER 7 Random-Access Text Files 123
About This Chapter 124
Random-Access Text Files 124
Record Length 125
Writing to a Record 126
Inside a Record 126
Reading From a Record 126
A Sample Program 127
Writing a Record 129
Reading a Record 131
Controlling the Program 134
The OPEN Command 135
The Options 136

The CLOSE Command 137
The Options 137
The WRITE Command 137
The Options 138
The End of File 138
The READ Command 139
The Options 139
The APPEND Command 140
The Options 140
For Example 141
The FLUSH Command 142
The Options 142
The POSITION Command 143
The Options 143

CHAPTER 8

EXEC: Control From a Text File 145
About This Chapter 146
EXEC Demonstration 147
Create an EXEC File Using BASIC 149
Printing the Commands to the File 149
An All-Purpose EXEC Maker Program 150
Listing a BASIC Program to a File 151
Use EXEC to Combine Programs 152
Machine Language to BASIC 152
The EXEC Command 153
The Options 153

CHAPTER 9

Binary Files 155

About This Chapter 156

This Chapter's Commands 157

Binary Files 159

Binary Addresses 160

The Memory Address Options: [,A#] [,E#] [,L#] 160

The File Position Option: [,B#] 162

The BRUN Command 162

The Options 163

The BLOAD Command 164

The Options 164

For Example 165

Using BLOAD With Non-Binary Files 166

High-Resolution Graphics With ProDOS 166

Installing Machine-Language Routines 167

The BSAVE Command 167

The Options 168

For Example 168

The PR# and IN# Commands 170

The Options 171

For Example 171

What PR# and IN# Really Do 172

ProDOS and the Monitor 173

Using a Clock/Calendar Card 174

System Programs 175

Starting Up a System Program 175

APPENDIX A

Summary of ProDOS 177
Features of ProDOS 178
Filenames 178
Pathnames 179
Syntax 180
Summary of the Options 181
ProDOS Commands in Programs 185
Filing Commands 186
CATALOG and CAT 186
PREFIX 187
CREATE 188
RENAME 188
DELETE 189
LOCK 189
UNLOCK 190
VERIFY 190
BASIC Program Commands 191
—(DASH) 191
RUN 191
LOAD 192
SAVE 193
Programming Commands 193
CHAIN 193
STORE 194
RESTORE 195
PR# 195
IN# 196
FRE 197
BYE 197

Text File Commands 197
OPEN 197
CLOSE 198
READ 199
WRITE 200
APPEND 201
FLUSH 201
POSITION 202
The EXEC Command 203
EXEC 203
Binary Commands 204
BRUN 204
BSAVE 204
BLOAD 206

APPENDIX B
DOS, ProDOS, and Applesoft 207
About This Appendix 208
DOS Disks and ProDOS Disks 208
Converting Files 209
The Differences Between DOS and ProDOS 210
File Organization and Names 211
DOS Commands That Went Away 211
FP and INT 212
INIT 212
MAXFILES 212
MON and NOMON 212

Improved DOS Commands 213
APPEND 213
BLOAD 213
BRUN 213
BSAVE 214
CATALOG 214
CHAIN 214
CLOSE 214
IN# and PR# 214
OPEN 214
POSITION 215
READ 215
RUN 215
WRITE 215
New ProDOS Commands 215
CAT 216
CREATE 216
FLUSH 216
PREFIX 216
STORE and RESTORE 216
—(DASH) 217
FRE 217
VERIFY 217
BYE 217
Changes to Applesoft 218
HIMEM 218
HGR, HGR2, and TEXT 218
INPUT 218
IN# and PR# 219
TRACE and NOTRACE 219
FRE 219

APPENDIX C

Error Messages 221

Handling Errors From Applesoft 222

Discussion of ProDOS Errors 225

RANGE ERROR (Code 2) 225

NO DEVICE CONNECTED (Code 3) 226

WRITE PROTECTED (Code 4) 226

END OF DATA (Code 5) 226

PATH NOT FOUND (Code 6 or Code 7) 228

I/O ERROR (Code 8) 228

DISK FULL (Code 9) 229

FILE LOCKED (Code 10) 229

INVALID PARAMETER (Code 11) 229

NO BUFFERS AVAILABLE (Code 12) 229

FILE TYPE MISMATCH (Code 13) 230

PROGRAM TOO LARGE (Code 14) 231

NOT DIRECT COMMAND (Code 15) 231

SYNTAX ERROR (Code 16) 231

DIRECTORY FULL (Code 17) 232

FILE NOT OPEN (Code 18) 232

DUPLICATE FILENAME (Code 19) 232

FILE BUSY (Code 20) 232

FILE(S) STILL OPEN (Code 21) 232

APPENDIX D

Extras 233
About This Appendix 234
Using the System Date and Time 234
Using TIME 234
Reading From ProDOS Directories 235
The Applesoft Programmer's Assistant (APA) 237
Overview 237
Starting APA 237
Automatic Line Numbering 238
Turning Off Automatic Line Numbering 239
Renumbering a Program 239
Putting a Program on Hold 242
Merging Two Programs Into One 242
Deleting Remarks From a Program 245
Displaying Control Characters 245
Suppressing Control Characters 245
Calculating a Program's Length 246
Producing a Cross-Reference Listing 246
Converting Decimal to Hex and Hex to Decimal 246
Clearing the APA Program From Memory 247

Glossary 249
Index 257

Figures, Tables, and Programs

CHAPTER 1

Introduction

Figure 1-1 The ProDOS Title Screen 4
Figure 1-2 /EXAMPLES Screen 5
Figure 1-3 The HELP Selection Screen 8
Figure 1-4 The CATALOG Help Screen 10

CHAPTER 2

Files and Commands

Figure 2-1 Files in a Directory 15
Figure 2-2 The Main Files on /EXAMPLES 17
Figure 2-3 The Structure of a ProDOS Pathname 19
Figure 2-4 A Sample Directory Structure 20
Table 2-1 The Prefix and Pathnames 21
Table 2-2 How ProDOS Forms a Pathname 26

CHAPTER 3

Using Files

Figure 3-1 Using Files 30
Figure 3-2 CAT and CATALOG 31
Figure 3-3 A Catalog of the /EXAMPLES Disk 35
Figure 3-4 A Catalog of /EXAMPLES/DIRECTORY 36
Table 3-1 The File Type Abbreviations 37
Table 3-2 The Subtype Column 38
Table 3-3 The File Type Abbreviations 43

CHAPTER 4

BASIC Programs in Files

Figure 4-1 BASIC in Files 54

CHAPTER 5

Programming With ProDOS

Figure 5-1 CHAIN 65
Figure 5-2 STORE and RESTORE 66
Figure 5-3 PR# and IN# 67
Figure 5-4 The Startup Process 70
Table 5-1 Memory Locations for Error Handling 74
Program APPLESOFT STARTUP 72
Program ONERR.DEMO 75
Program PART1 79
Program PART2 79
Program E.S.P. 82

CHAPTER 6

Text in Files

Figure 6-1 Printing to a Scroll 94
Figure 6-2 Printing to a Notebook 97
Table 6-1 Printing to a Text File 103
Table 6-2 Reading From a Text File 104
Program LISTSELF 101
Program MAKE.FRUIT 105
Program GET.FRUIT 106
Program CONJUGATE 107
Program CONJUGEATEN 107
Program CONJUGEAT 108
Program MAKE.TEXT 111
Program GET.TEXT 112
Program APPEND.TEXT 112

CHAPTER 7

Random-Access Text Files

Figure 7-1 Sequential and Random-Access Text Files 125
Figure 7-2 Five Addresses in the File 127
Figure 7-3 Writing an Address to Record Five 129
Figure 7-4 Reading an Address From Record Five 131
Program ADDRESS 128

CHAPTER 8

EXEC: Control From a Text File

Program AWAY 150
Program MAKE.DOIT 150
Program CAPTURE 151
Program MACHINE LANGUAGE POKER 152

CHAPTER 9

Binary Files

Figure 9-1 BRUN, BLOAD, and BSAVE 157
Figure 9-2 PR# and IN# 158
Figure 9-3 Memory Address Options A#, E#, and L# 161
Figure 9-4 ProDOS Date and Time Locations 174

APPENDIX A

Summary of ProDOS

Table A-1 ProDOS Command Options 180

Table A-2 The File Type Abbreviations 185

APPENDIX B

DOS, ProDOS, and Applesoft

Table B-1 File Conversion 210

APPENDIX C

Error Messages

Table C-1 Error Message Formats 222

Table C-2 ProDOS Error Codes 223

Table C-3 Errors by ProDOS Command 224

Table C-4 Applesoft Error Codes 225

APPENDIX D

Extras

Table D-1 Directory Line Composition 236

Preface

The BASIC Language and Disk Operating Systems

You may already have discovered that you can use the Applesoft BASIC programming language that is built into your Apple II computer to write and run BASIC programs even if you have no disk drive connected to your computer. But—without a disk drive and an operating system, you can't save the programs you write.

A disk operating system is software that serves as a housekeeper for the information stored on disks. ProDOS™ is Apple's Professional Disk Operating System. It allows you to place information on disks, rearrange the information that is already on the disks, and retrieve information from disks.

The manual you are now reading takes you one step further: it shows you, among other things, how to write programs that use ProDOS to create and read data files on a disk.

The Parts of ProDOS

The ProDOS operating system is sold as part of various packages.

The disk labeled *ProDOS BASIC Programming Examples* is the disk that came with *this* manual. It contains ProDOS, as well as a number of other programs that are explained in this manual.

Your Apple IIc package may have included the *System Utilities* disk and the *System Utilities* manual. The *System Utilities* disk contains ProDOS, as well as several utility programs. The *System Utilities* manual explains how to use the *System Utilities* disk.

If you have an Apple IIe or II Plus, you may have a manual called the *ProDOS User's Manual* and a disk called the *ProDOS User's Disk*. These came either with your disk drive or in a *ProDOS User's Kit*. The *ProDOS User's Disk* contains ProDOS, as well as several utility programs. The *ProDOS User's Manual* explains how to use the *ProDOS User's Disk*.

The *ProDOS User's Disk* and the *System Utilities* disk perform comparable functions: they both let you format new disks, copy files and volumes, set the time and date, and so on. Many exercises and examples in this book call for you to use one or the other of these disks. For brevity's sake, they're sometimes referred to as the ProDOS utilities disks.

There is another ProDOS manual that you may want to know about. The *ProDOS Technical Reference Manual* explains what makes ProDOS tick. An experienced programmer can use the information in this manual to write machine-language programs that use ProDOS.

By the Way: The catalog of your *Examples* disk may show dates and times that are not the same as those pictured in this manual. Don't worry—it is just to show you what these catalogs look like.

Requirements for Using ProDOS

All Apple IIe and IIc computers can use ProDOS. To use ProDOS, an Apple II or II Plus must have at least 64K of random-access memory (RAM) and Applesoft BASIC in read-only memory (ROM). (If your Apple II has Integer BASIC in ROM, you must replace the Integer BASIC ROM with an Applesoft ROM.) Your Apple II must also have at least one disk drive attached (or built in, in the case of the Apple IIc).

Before using this manual, you should be familiar with your Apple II, and with the use of Applesoft BASIC. One way to become familiar with BASIC is to work your way through the *Applesoft Tutorial*. If you have never used files that are grouped into directories, read (in either the *ProDOS User's Manual* or the *System Utilities* manual) about files, filenames, and pathnames.

Important

If you are experienced with DOS, the predecessor to ProDOS, read Chapters 1 and 2 of this manual and then skip to the appendix that describes the differences between DOS and ProDOS.

Important Note for Apple IIc Owners

The Apple II, II Plus, and IIe computers are characterized by expansion slots, narrow sockets inside the computer where peripheral cards can be installed. A peripheral card is a removable printed circuit board, somewhat like a small computer within a computer, that expands or modifies a computer's capability. It might, for example, allow the connection of a printer, disk drive, modem, mouse, or other peripheral device. Or it might add memory or other features to the computer system.

To maintain compatibility, the Apple IIc's ports have the same firmware entry points as the equivalent Apple IIe slots. Many features that must be *added* to the Apple IIe via the slots are built into the Apple IIc. So, even though the Apple IIc has no slots, it is the equivalent of an Apple IIe with five peripheral cards installed:

- serial printer interface (slot 1)
- serial communications interface (slot 2)
- 80-column display capability and 64K bytes of extra memory (slot 3)
- mouse interface (slot 4)
- controller for two disk drives (slot 6) The built-in drive is drive 1 and a second disk drive, if present, is considered to be drive 2.

In this book, you'll see a number of references to slots and drives. For example, if a printer command requires that you indicate the printer's slot, use *slot 1*, even though there is no actual slot.

Several commands allow you to specify the location of disks by slot and drive. Even though your Apple IIc manuals call it simply the built-in drive, use *slot 6, drive 1*. If you have a second disk drive connected to your Apple IIc, it is *slot 6, drive 2*.

Chapter 5 tells you that you can start up from the disk drive connected to slot 6 by typing `PR#6`—this works with an Apple IIc as well as it does with the computers that have real slots.

This Manual's Organization

This manual is organized to be useful to as many people as possible. Each chapter begins with a section describing the topics it covers. If a chapter introduces new ProDOS commands, it also has a section called "This Chapter's Commands." Read this section carefully; you can use its command summary to find commands with which you are unfamiliar.

Notice that each command description has three sections. The first section tells you why you would want to use the command, shows its form, or syntax, and gives a brief example. The second section, "The Options," gives a precise definition of the different abilities of the command. The third section, "For Example," is a hands-on example of the command. If you learn best by doing, try these examples.

Here is a quick summary of the chapters and appendixes in this manual.

Chapter 1:	What you should know before using ProDOS.
Chapter 2:	What files are and how they are named. What ProDOS commands are and how they are structured.
Chapter 3:	How to keep track of and manipulate the files on a disk.
Chapter 4:	How to use programs that are stored on a disk.
Chapter 5:	How to write BASIC programs that use ProDOS commands.
Chapter 6:	How to write BASIC programs that use sequential text files.
Chapter 7:	How to write BASIC programs that use random-access text files.
Chapter 8:	How to make files control the operation of your Apple computer.
Chapter 9:	How to use binary programs and files, and the Monitor. How to use a clock/calendar card.
Appendix A:	A summary of ProDOS.
Appendix B:	The differences between DOS and ProDOS.
Appendix C:	Error messages.
Appendix D:	The Applesoft Programmer's Assistant, and other extras.

Watch for These

Look for these throughout the manual:

By the Way: Text set off in this manner presents sidelights or interesting pieces of information.

Important Text set off in this manner—and with a tag in the margin—presents important information.

▲Warning Warnings like this indicate potential problems or disasters.

Chapter 1

Introduction

Before You Start

This book assumes that you already have some experience using your Apple II. Since this is a book about programming in BASIC, you should be familiar with the material in the *Applesoft Tutorial.*

If your computer is an Apple II Plus or IIe, you should also be familiar with Chapter 2 of the *ProDOS User's Manual.* If your computer is an Apple IIc, you should also be familiar with Chapters 2 and 3 of your *System Utilities* manual.

Your computer should be set up properly with at least one disk drive (the Apple IIc's built-in drive is sufficient).

As you read this book, you should have these things close at hand:

- One of the ProDOS utilities disks—*System Utilities* or the *ProDOS User's Disk*
- The manual that explains how to use your utilities disk—*ProDOS User's Manual* or *System Utilities* manual
- The disk labeled *ProDOS BASIC Programming Examples* that came with this book
- At least one blank disk
- The *Applesoft Tutorial* (optional).

Make a Spare Copy of Examples Disk

While you learn to work with ProDOS, you are asked to do many things with your *Examples* disk. Because the *Examples* disk is write-protected—that is, it does not have a write-enable notch—you can't write information on it.

In your *ProDOS User's Manual* or *System Utilities* manual, find the instructions for backing up disks. Then make a backup copy of your original *Examples* disk, naming the new disk /EXAMPLES. ProDOS requires that volume names begin with a slash, but doesn't care whether you use uppercase or lowercase letters. With a felt-tip pen, write your name and */EXAMPLES* on the disk label. Put the original in a safe place; someday you may want to copy it again.

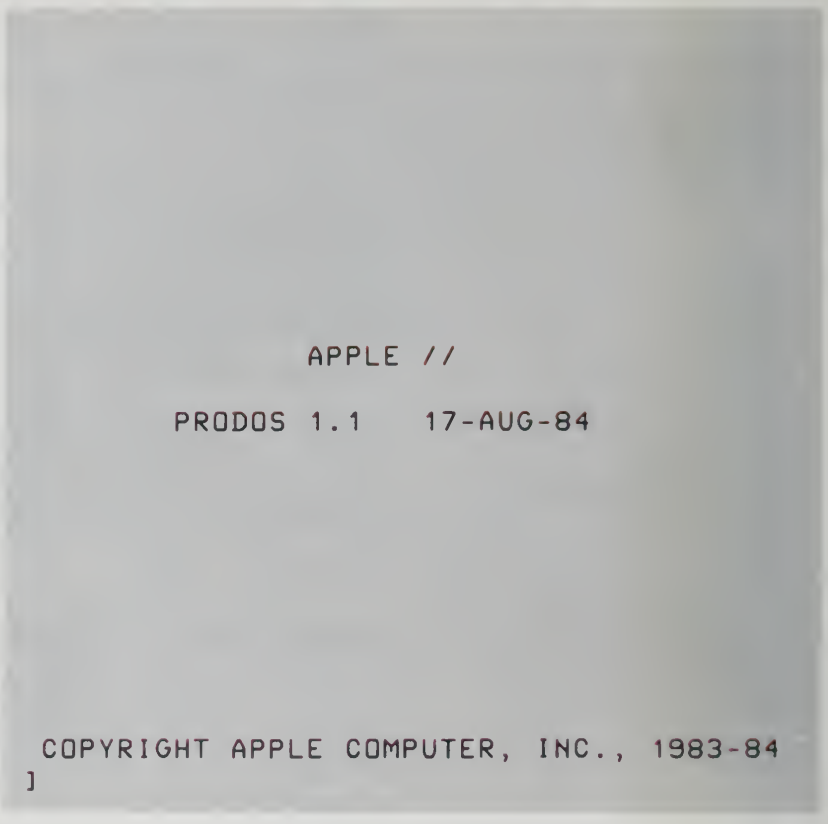
Be Careful

Even if someone else has already made a copy of /EXAMPLES, it is best to make one of your own. Some of the examples in this book ask you to change the files on the disk. If someone has already done these examples, the disk will not be in its original form.

Starting ProDOS BASIC

Place your copy of the /EXAMPLES disk in drive 1 (or in your Apple IIc's built-in drive), and close the drive door. Start up your computer in the usual way. The display screen will soon look something like Figure 1-1.

Figure 1-1. The ProDOS Title Screen



This tells you that ProDOS is now in the computer's memory. Any disk that contains a ProDOS program shows this display when it starts up.

After a few moments more, you see a display that looks something like Figure 1-2. Its exact appearance depends on the type of Apple II system you are using, and the kinds of peripheral cards that are installed in your system. If you have an Apple IIc, your computer contains the equivalent of five peripheral cards, even though it has no *slots*.

Figure 1-2. /EXAMPLES Screen

```
****************************************
*                                      *
*  PRODOS BASIC PROGRAMMING EXAMPLES   *
*                                      *
* COPYRIGHT APPLE COMPUTER, INC. 1984  *
*                                      *
****************************************
STARTUP DISK: /EXAMPLES/

YOUR Apple //e HAS:

    64K OF RANDOM ACCESS MEMORY

    APPLESOFT IN ROM

    SLOT 1: SILENTYPE
    SLOT 2: EMPTY
    SLOT 3: 80-COLUMN CARD
    SLOT 4: CLOCK
    SLOT 5: PROFILE
    SLOT 6: DISK DRIVE
    SLOT 7: EMPTY
```

The startup disk name.

The type of Apple II.

The amount of RAM in your Apple II.

Applesoft BASIC is in read-only memory.

The contents of each of your Apple II's slots.

This display contains a lot of information. It tells you that the ProDOS program was brought into memory from the disk named /EXAMPLES. This display also describes your system: how much memory it has, which versions of BASIC you can use, and the type of peripheral card installed in each of your Apple II's peripheral connector slots.

The Startup Process

When you start up your Apple IIc, it tries to read information from the built-in drive. When you start up other Apple II's, they try to read information from drive 1 of the disk controller card in the highest numbered slot (usually slot 6) inside the computer. If the computer finds a ProDOS startup disk, the ProDOS program is brought into memory. You see the two displays described above, followed by the BASIC prompt:

```
]
```

When you type a few lines of BASIC, your Apple II seems to behave just as it would without ProDOS or the disk drive. The startup process actually added the ProDOS commands to the BASIC commands to which you are accustomed. Several new commands have been added, and several of the old commands have been enhanced.

▲Warning Even though the ProDOS commands look like BASIC commands (as you will see), they do not always follow the same rules.

Other Ways to Start ProDOS BASIC

On an Apple IIe or IIc, you can always start the system when the power is already on, by pressing the three keys—(Ó), (CONTROL), and (RESET)—all at the same time, and then releasing them.

On any Apple II computer, you can start the system by turning the computer's power off and then on again.

When you see one of the prompts (`]` or `*`), you can usually restart the system by typing

```
PR# 6
```

On all Apple II's, except the IIc, if the disk controller card is installed in another slot, replace 6 with the slot's number.

If you see the monitor prompt (`*`), and `PR# 6` doesn't work, try typing

(6) (CONTROL)-(P)

and pressing (RETURN)

Important

These commands are explained later. For now, remember that using them is a good way to start over if things seem to be hopelessly confused. Beware, however—though they don't affect anything on disk, these commands make everything that is in memory disappear.

The HELP Command

Loading HELP

If you are using ProDOS and can't quite remember the exact form of a command, you can add the HELP command. With /EXAMPLES in a disk drive, type

```
- /EXAMPLES/HELP
```

The disk spins for a moment, while HELP is loaded into memory. You need to type this command only once. The HELP command remains in memory until you turn off your computer or use another program.

Using HELP

When HELP is in memory and the /EXAMPLES disk is in a drive, you can get help with ProDOS commands by simply typing

```
HELP
```

You will then see a list, shown in Figure 1-3, of the ProDOS commands you can get help with.

Figure 1-3. The HELP Selection Screen

```
                HELP (a user-added command)

To use, type:    HELP XXX  (XXX's below)

Explanatory:     HELP, SYNTAX, BINARY,
                 FILE

Using Files:     CAT, CATALOG, PREFIX,
                 CREATE, RENAME, DELETE,
                 LOCK, UNLOCK

BASIC in Files:  - , LOAD, RUN, SAVE

Programming:     CHAIN, STORE, RESTORE,
                 PR#, IN#

Text in Files:   OPEN, CLOSE, WRITE,
                 READ, APPEND, POSITION,
                 FLUSH, EXEC

Binary Files:    BLOAD, BRUN, BSAVE
```

Where Explained	
Explanatory displays. See Chapters 1, 2, 9.	
See Chapter 3.	
See Chapter 4.	
See Chapter 5.	
See Chapters 6, 7, 8.	
See Chapter 9.	

The commands are grouped on the screen in the same way they're grouped in the chapters in this book. Within the groups, the commands are listed in the same order in which they're discussed in the chapter.

Using HELP Plus a Command Name

If, instead of typing just **HELP**, your type **HELP** followed by the name of a command, you'll see a display of information about that command.

For Example

If your Apple II is turned off, make sure /EXAMPLES is in drive 1, then turn on the computer.

As soon as the prompt character appears on the display, type

```
- /EXAMPLES/HELP
```

When the BASIC prompt returns, type

```
HELP
```

and you will see the list of commands shown in Figure 1-3. If you want to see a typical help screen, type

```
HELP CATALOG
```

and you see the display shown in Figure 1-4.

Figure 1-4. The CATALOG Help Screen

```
                CATALOG          (Imm & Def)

  Show an 80-column directory listing

Form:  CATALOG  [pn] [,S#] [,D#]

Example:  CATALOG /BUDGET/JAN
          CATALOG, S6, D2

The first example lists the files in
the directory /BUDGET/JAN; the second
lists files in volume directory S6, D2.

This command shows the same items as
CAT.  In addition, it shows the date
the file was created, the logical
length of the file, and some subtype
information.

See HELP CAT.
```

At the top of the display is the name of the command, **CATALOG**, followed by the message **(Imm & Def)**. This message tells you that the CATALOG command can be used in immediate mode (as a command typed from the keyboard), and in deferred mode (as a line in a program).

Next is a line that describes what the command does, one that shows the form (syntax) of the command, two lines of examples, and then an explanation of the command syntax. All the abbreviations shown in the display will be much clearer after you finish Chapter 2.

All the help screens use this same format, with some variations due to the requirements of each individual command.

Installing HELP on Other Disks

For the HELP command to work, the two files named HELP and HELPSCREENS must be on the same disk. To copy them onto a disk other than /EXAMPLES, use the *ProDOS User's Disk* or *System Utilities* disk. This is explained in more detail later.

Deleting HELP

You can remove the HELP command, thus freeing up some computer memory, by typing

```
NOHELP
```

This should be necessary only if you are writing an extremely large BASIC program. If you try to use HELP commands after using NOHELP, you'll see the message `SYNTAX ERROR`.

ProDOS and DOS

DOS is the disk operating system that was used with Apple II computers before ProDOS was introduced. Because there are many programs available that require DOS, you may need to know some of the differences between ProDOS and DOS.

DOS can be used only with 5¼-inch single-sided flexible disk drives such as Disk II drives and their equivalents. ProDOS, on the other hand, is not limited to Disk II drives—it can also be used with mass storage devices, such as the ProFile hard disk drive.

DOS can't read information that was placed on a disk by ProDOS, and ProDOS can't read information that was placed on a disk by DOS. However, in some cases you can use the *ProDOS User's Disk* or the *System Utilities* disk to convert information from one format to the other.

Appendix B, which describes the differences between ProDOS and DOS, explains how to convert a program from one format to another.

This means that you can use your existing DOS programs only on a DOS-formatted disk, unless they can be converted from DOS format to ProDOS format. As a general rule, programs you buy can't be converted, but programs you write yourself *can* be. If a program uses only BASIC and DOS commands, it can be converted. If it does any tricky PEEKs and POKEs, or if it uses machine language at all, then it probably cannot be automatically converted.

Chapter 2 Files and Commands

About This Chapter

The first part of this chapter is about files: how ProDOS arranges files on a disk, and how files are named.

The second part of the chapter explains how ProDOS commands are structured. These are the commands that let you make use of your files.

When you come to sections entitled "For Example," *try* the examples. (For the examples to work, ProDOS must be running, as explained in Chapter 1, and the /EXAMPLES disk must be in a drive.)

Disks

The *ProDOS User's Manual* and the *System Utilities* manual explain how to format disks.

The purpose of ProDOS is to let you use the information on disks. But before ProDOS can use a disk, the disk must be prepared for use, or **formatted.**

See Appendix B for more details on DOS and ProDOS.

You'll find it convenient always to have an adequate supply of empty ProDOS-formatted disks on hand, and to mark each disk so that you know that it is ProDOS-formatted. ProDOS can't use disks of other formats (DOS, Apple II Pascal). Data on disks formatted by Apple III SOS can be used by ProDOS (although the programs on them cannot).

Files

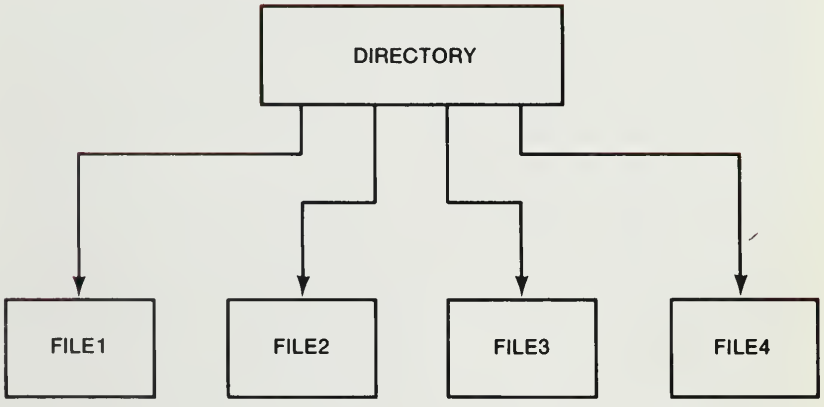
ProDOS lets you organize information into units of disk storage known as files. Files can contain numbers, phone lists, letters, pictures, programs, or any other type of information that your Apple II can use.

When a file is placed on a disk, it is assigned a name and a type. When you want access to the information stored in a particular file, refer to that file by its name. The file's type indicates the kind of information that the file contains. For example, there are text files, program files, and directory files.

Directory Files

Instead of containing a program or text, a directory file contains a list of other files and their locations on the disk.

Figure 2-1. Files in a Directory



The directory shown in Figure 2-1 contains four files: FILE1, FILE2, FILE3, and FILE4. Each of these files can be of any file type; thus all, some, or none of them could be directory files. You can have up to 64 levels of directories on a disk, but it's difficult to use more than five or six levels.

Volume Directory Files

A volume directory is the main directory file for the entire disk.

When you use ProDOS to format a disk, a special type of directory file is automatically placed on the disk. It is called a **volume directory**, and it is the main directory file for the entire disk.

A ProDOS volume directory

- is on every ProDOS-formatted disk.
- can contain up to 51 files.
- has a name, which you specify when you format the disk. This name, being associated with the entire contents of the disk, is also the disk's name. You should place it on the disk's label.
- is the only file that you cannot create using the CREATE or MAKE DIRECTORY command.
- is the only file that you cannot remove from a disk (although you can remove all the files in it). It is also the only file that you cannot protect using the LOCK command.

The /RAM Directory

If you have an Apple IIc, or an Apple IIe with an Extended 80-Column Text Card, your computer has 128K of RAM. ProDOS creates a volume directory file in the alternate 64K of this RAM. This volume is named /RAM, and it can be used just like a small disk. Unlike a disk, however, the information in /RAM disappears when you turn off your computer. You should use this volume only for temporary storage of information. An example of the use of /RAM is given in the following section.

For Example

To see the contents of the volume directory of the *Examples* master disk, type the ProDOS command

CAT is the short version of CATALOG.

```
CAT /EXAMPLES
```

and the display shown in Figure 2-2 appears.

By the Way: Your disk catalog won't look exactly like the one pictured here. That's OK.

Figure 2-2. The Main Files on /EXAMPLES

```
/EXAMPLES
 NAME            TYPE   BLOCKS   MODIFIED

 PRODOS           SYS       29   1-SEP-83
 BASIC.SYSTEM     SYS       21   1-SEP-83
 STARTUP          BAS        7   1-SEP-83
 HELP             BIN        1   1-SEP-83
 HELPSCREENS      TXT       59   1-SEP-83
 DIRECTORY        DIR        1   1-SEP-83
 PRACTICE         DIR        1   1-SEP-83
 PROGRAMS         DIR        3   1-SEP-83
 DATA             DIR        1   1-SEP-83
 EXTRAS           DIR        1   1-SEP-83
 LISTING.EXEC     TXT        1   1-SEP-83

 BLOCKS FREE:  25      BLOCKS USED:  255
```

Some of these files are directory files: they contain the names and locations of other files on the disk. You can recognize a directory file by the abbreviation **DIR** to the right of its filename in the catalog.

If you have an Apple IIc, or an Apple IIe with an Extended 80-Column Text Card, type the ProDOS command

```
CAT /RAM
```

Chapter 4 contains more information about the use of /RAM.

ProDOS displays information similar to that shown above, but with no filenames listed. The directory /RAM is empty.

Filenames

Several ProDOS commands cause files to be created; each requires that you assign the file a filename.

A ProDOS filename

- is composed of up to fifteen characters. The first must be a letter; the rest can be any combination of uppercase or lowercase letters (A-Z), digits (0-9), and periods (.). Lowercase letters are automatically converted to uppercase, thus *A* and *a* are equivalent.
- must be unique within its directory. There can be files by the same name in different directories.

Some Legal Filenames

Here are a few legal filenames (assuming there aren't already files of the same name in the same directory).

A.LONG.FILENAME	Longest possible name
Z	Shortest possible name
A.1DERFUL.NAME	Has letters, number, and periods

Note | Although you can't use spaces in filenames, you *can* use periods to separate words within a filename.

Some Illegal Filenames

Here are some illegal filenames and the reasons you can't use them.

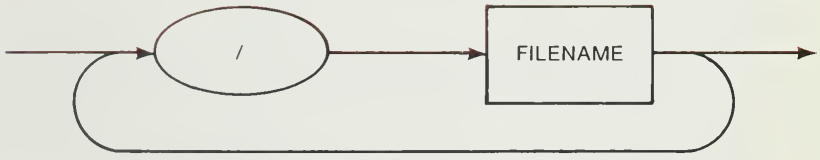
3.BLIND.MICE	Begins with a number
.PRINTER	Begins with a period
SPACE RACE	Contains a space
BOOP,BETTY	Contains a comma
PEANUT.BUTTER.AND.PICKLES	Too many characters

Pathnames

To find a file, ProDOS must know the path (from the disk's volume directory to the file) that it must follow to get to the file. The entire path, from the volume directory to the file, is called the file's **pathname**. For example, the pathname to a file named STAND in a directory named LAST in the volume directory CUSTER is /CUSTER/LAST/STAND.

Figure 2-3 represents the structure of a pathname.

Figure 2-3. The Structure of a ProDOS Pathname



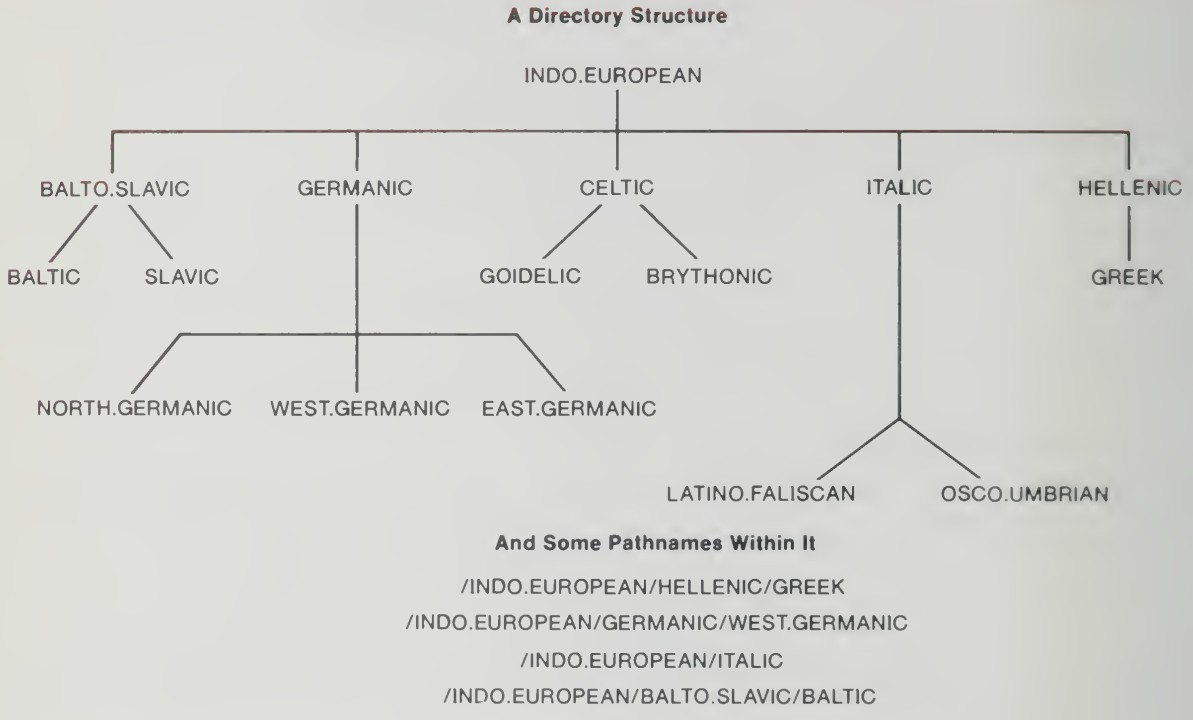
A ProDOS pathname

- begins with a slash (/) followed by a volume directory filename
- is a series of filenames, separated by slashes
- is no more than 64 characters long, including slashes.

Figure 2-4 displays the directory structure of an imaginary disk containing files that describe part of the Indo-European family of languages. Below it are a few of the many valid pathnames within this directory structure.

In this example, /INDO.EUROPEAN is the name of the disk containing these files, and is also the name of the disk's volume directory.

Figure 2-4. A Sample Directory Structure



The Prefix and Partial Pathnames

It takes time to type an entire pathname every time you use a file. By setting the prefix, a pathname that indicates a directory file, you can refer to files in that directory, or to files that can be reached through that directory, by using their **partial pathnames**.

A partial pathname is a file's pathname with the prefix removed from the front of it.

Table 2-1 shows the relationship between pathname, prefix, and partial pathname.

Table 2-1. The Prefix and Pathnames

You Want ProDOS to Find	Current Prefix Is	You Should Type
/CAROL/GHOSTS/XMAS.PAST	/CAROL/	GHOSTS/XMAS.PAST
/CAROL/GHOSTS/XMAS.FUTURE	/CAROL/GHOSTS/	XMAS.FUTURE
/MY.DISK/GAMES	/YOUR.DISK/	/MY.DISK/GAMES

In the third column of Table 2-1, GHOSTS/XMAS.PAST and XMAS.FUTURE are both partial pathnames; a full pathname is formed by adding the current prefix. In the third example you want to use /MY.DISK/, but the prefix is set to /YOUR.DISK/ . In cases like this, you must use the file's full pathname (or change the value of the prefix).

Make sure you understand the examples in Table 2-1. Once you do, you will never have trouble with pathnames, partial pathnames, or prefixes.

A ProDOS partial pathname

- is a filename, or a series of filenames separated by slashes
- is a pathname minus the current prefix
- is no more than 64 characters long, including slashes.

To set the prefix, type **PREFIX**, followed by a space and a valid directory name.

For Example

Before referring to several files that are in the /EXAMPLES/PROGRAMS directory, you can use the command

```
PREFIX /EXAMPLES/PROGRAMS
```

Now, you can look at the contents of the /EXAMPLES/PROGRAMS directory with the command

```
CAT
```

As long as the prefix remains /EXAMPLES/PROGRAMS, you can see a list of the files in that directory without having to type **CAT /EXAMPLES/PROGRAMS**.

The General Form of ProDOS Commands

This manual describes all the possible forms of each ProDOS command by presenting a one-line description of the command. This one-line description is called the command's general form, or **syntax**, and it looks something like this:

command [pn] [,S#] [,D#]

The word *command* represents any of the ProDOS commands. The expressions [pn], [,S#], and [,D#] are the command's options. There are many options other than these three.

The three options shown above are called the pathname, slot, and drive options. Together they determine the name and location of the file to be accessed. You can specify a file on any of your disks using only the pathname option. The slot and drive options give you additional control in accessing files.

Here is a specific instance of the CATALOG command that uses the [pn], [,S#], and [,D#] options:

```
CATALOG BOOKS, S6 , D1
```

This command tells ProDOS to display the files contained in the BOOKS directory (in this case *pn* is replaced by a partial pathname), which is in the volume directory of the disk in slot 6, drive 1. (In an Apple IIc, the built-in drive is considered to be slot 6, drive 1.) Note the use of commas in the above example. Commas separate the options; you can put spaces before or after the commas if you wish.

Options

Sometimes an option is shown with brackets around it, sometimes not.

If an option is shown without brackets around it, it cannot be omitted—you must use that option each time you use the command, and you must use it in the order shown by the command's syntax.

An option shown within brackets may be included or omitted, depending on what you want the command to do. Bracketed options can be used in any order, except that pathname must precede any other options used.

▲Warning When you use an option, never type the square brackets; they are only there to tell you that the option is not required.

The characters within the brackets do three things: the comma separates an option from its predecessor, the capital letter identifies which option you are using, and whatever is after the capital letter (usually #) stands for the value you can give that option.

The letters *pn* should be replaced by a pathname or partial pathname, as explained below, and # should be replaced by an integer. The value of # can be either a decimal integer or a hexadecimal integer.

Note An additional notation is used later in the manual. Two options separated by a vertical bar, |, are alternates. Use one or the other, not both. You can read a bar as the English word *or*.

Hexadecimal Notation

You are not required to use hexadecimal numbers. The integer in an option, represented by #, can be expressed in hexadecimal notation by preceding the hexadecimal digits with a dollar sign. For example, the decimal integer 254 can be expressed in hexadecimal notation as $FE.

The Pathname Option—[pn]

[pn] This option indicates to ProDOS the name of the file that you want to use. You can replace pn with a pathname or a partial pathname.

If you use a pathname, ProDOS looks for the file with that pathname.

If you use a partial pathname without the [,S#] and [,D#] options, ProDOS looks for the file having the pathname formed by the partial pathname added to the prefix. If the prefix is empty, the name of the volume indicated by the last used values of [,S#] and [,D#] is used in place of the prefix.

Note You can always access a file by using just its pathname. The [,S#] and [,D#] options are primarily for DOS compatibility. They are also useful if you don't remember a disk's name.

For Example —[pn]

Here are a few ProDOS commands that use the pathname option:

```
CAT /EXAMPLES
PREFIX /EXAMPLES/PROGRAMS/
RUN WHIZBOOM
```

Did you try these commands? If not, try them in the order presented. How they work is explained in later chapters.

The Slot Number Option —[,S#]

[,S#] Include this option to tell ProDOS the slot that connects the disk drive you want to access. Replace # with a slot number from 1 to 7.

When you use this option, the value specified by # becomes the default slot number.

If you use this option without the drive option, drive 1 is assumed.

If this option is used after a pathname, ProDOS first looks for that path in the indicated slot. If this option is used after a partial pathname, ProDOS forms a pathname by adding the volume name of the indicated volume to the partial pathname (the prefix is ignored).

The Drive Number Option —[,D#]

[,D#] Include this option to tell ProDOS the drive that contains the disk you want to access. Replace # with drive number 1 or 2.

When you use this option, the value specified by # becomes the default drive number. If you use the drive number option without a slot number option, ProDOS looks for the drive in the default slot.

If this option is used after a pathname, ProDOS first looks for that path in the indicated drive. If this option is used after a partial pathname, ProDOS forms a pathname by adding the volume name of the disk in the indicated slot and drive to the partial pathname.

▲Warning If you use the slot number and drive number options to indicate a drive that is not there, you will get the `NO DEVICE CONNECTED` error message.

For Example —[,S#] [,D#]

Here are some ProDOS commands that use the [,S#] and [,D#] options. Try them in order now with the /EXAMPLES disk in drive 1.

```
CAT PROGRAMS,S6,D1
```

displays the files in the PROGRAM directory of /EXAMPLES, the disk in slot 6, drive 1.

```
PREFIX , D1 , S6
```

sets the prefix to the name of the volume directory, slot 6, drive 1.

```
RUN /EXAMPLES/PROGRAMS/WHIZBOOM ,S6 ,D2
```

checks first in drive 2 for the /EXAMPLES volume, then in other drives until it finds /EXAMPLES.

In Summary

Table 2-2 shows the pathname that ProDOS seeks for each possible combination of the pathname, slot, and drive options.

Table 2-2. How ProDOS Forms a Pathname

[pn]	[,S#]	[,D#]	Pathname Sought
—	—	—	See command description
ppn	—	—	pn = prefix + ppn †
ppn	+	+	pn = vn + ppn
ppn	+	—	pn = vn + ppn ‡
ppn	—	+	pn = vn + ppn ††
pn	—	—	pn = pn
pn	+	+	pn = pn

Key:	+	= option used
	—	= option not used
	pn	= pathname
	ppn	= partial pathname
	vn	= volume name of disk at S#, D#

† If the prefix is empty, the last used values of S# and D# are used to determine a volume name.

‡ When only S# is given, drive 1 is assumed.

†† When only D# is given, the last value of S# is used.

Chapter 3 Using Files

About This Chapter

This chapter describes the ProDOS commands that let you keep track of and manipulate the files on your disks. You can use these commands

- to see what information is on a disk
- to create more room on a disk by throwing away obsolete information
- to change the name of some information so that it is easier to locate
- to protect some information from being accidentally destroyed.

Because you issue the commands described in this chapter from the keyboard more frequently than from within programs, they are described here before the chapter that explains the technique for using ProDOS commands in programs. Remember that they can all be used in programs.

This Chapter's Commands

The ProDOS commands described in this chapter let you manipulate files. Each of them affects one file at a time. If you want to perform operations on several or all of the files on a disk, use your *ProDOS User's Disk* or *System Utilities* disk. The commands are summarized in the following list.

CATALOG: List all the files in a directory

Use this command to see, on the screen, a list of all the files in the directory you name, as well as other information about each file.

PREFIX: Set a directory to work in

Use this command to set the value of the prefix, that is, the pathname that is automatically added to the beginning of any partial pathname you use. You can also use it to display the prefix.

CREATE: Create a new directory (or other file)

Use this command to create a new file with a name and type that you specify. Although there are ways to create other file types, this is the only way you can create a new directory.

RENAME: Change a file's name

Use this command to change the name of a file, but you cannot use it to move the file from one directory to another.

DELETE: Remove a file from a disk

Use this command to remove a file from its directory. Once you delete a file, it is not possible to get it back.

LOCK: Protect a file from being changed or deleted

Use this command to protect a file from being accidentally changed or deleted. Once you lock a file, it cannot be renamed, deleted, or otherwise changed until it is unlocked.

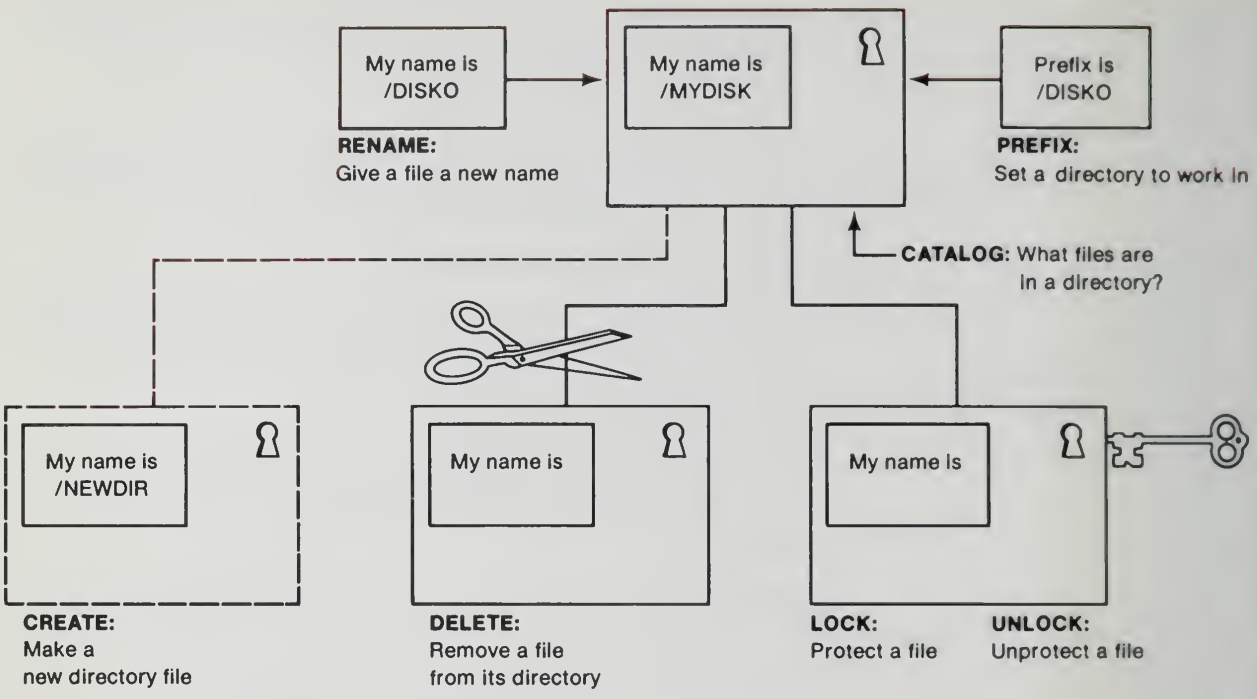
UNLOCK: Unprotect a locked file

Use this command so you can rename, delete or otherwise change a file that is locked.

VERIFY: Verify that a file exists

Use this command before creating a file, to be sure that another file with the same name doesn't already exist.

Figure 3-1. Using Files



The CAT and CATALOG Commands

To see the names and other information about all the files in any directory, use one of the two forms of the CATALOG command:

CAT [pn] [,S#] [,D#]	40-column display
CATALOG [pn] [,S#] [,D#]	80-column display

CAT, the shorter version of the command, gives you forty columns of information, including each file's name, type, size, and date of last change. CATALOG gives you eighty columns of information, adding the date that the file was created, and some technical information. If you display the 80-column list on a 40-column wide screen, each entry in the list takes up two lines. Figure 3-2 is a comparison of the four displays that can be generated by these commands.

Figure 3-2. CAT and CATALOG: 40-column CAT

```
/EXAMPLES
 NAME            TYPE  BLOCKS  MODIFIED

 PRODOS           SYS      29   1-SEP-83
 BASIC.SYSTEM     SYS      21   1-SEP-83
 STARTUP          BAS       7   1-SEP-83
 HELP             BIN       1   1-SEP-83
 HELPSCREENS      TXT      59   1-SEP-83
 DIRECTORY        DIR       1   1-SEP-83
 PRACTICE         DIR       1   1-SEP-83
 PROGRAMS         DIR       3   1-SEP-83
 DATA             DIR       1   1-SEP-83
 EXTRAS           DIR       1   1-SEP-83
 POSTAGE.RATES    BAS       8   1-SEP-83
 RENUMBER         BAS

BLOCKS FREE:  107      BLOCKS USED:  173
```

Figure 3-2—Continued. CAT and CATALOG: 80-column CAT

```
/EXAMPLES
 NAME            TYPE  BLOCKS  MODIFIED

 PRODOS           SYS      29   1-SEP-83
 BASIC.SYSTEM     SYS      21   1-SEP-83
 STARTUP          BAS       7   1-SEP-83
 HELP             BIN       1   1-SEP-83
 HELPSCREENS      TXT      59   1-SEP-83
 DIRECTORY        DIR       1   1-SEP-83
 PRACTICE         DIR       1   1-SEP-83
 PROGRAMS         DIR       3   1-SEP-83
 DATA             DIR       1   1-SEP-83
 EXTRAS           DIR       1   1-SEP-83
 POSTAGE.RATES    BAS       8   1-SEP-83
 RENUMBER         BAS


BLOCKS FREE:  107      BLOCKS USED:  173
```

Figure 3-2—Continued. CAT and CATALOG: 40-column CATALOG

```
/EXAMPLES
 NAME            TYPE  BLOCKS  MODIFIED
       CREATED           ENDFILE SUBTYPE

 PRODOS          SYS       29   1-SEP-83
 0:00  27-JUL-83 17:16     14336
 BASIC.SYSTEM    SYS       21   1-SEP-83
 0:00   5-AUG-83 21:04     10240
 STARTUP         BAS        7   1-SEP-83
11:41  22-JUL-83  0:00      2750
 HELP            BIN        1   1-SEP-83
11:27  22-JUL-83 11:27         54
 HELPSCREENS     TXT       59   1-SEP-83
 0:00  22-JUL-83  0:00 12358656
 DIRECTORY       DIR        1   1-SEP-83
 0:00  28-MAR-83  0:00       512
 PRACTICE        DIR        1   1-SEP-83
 0:00  18-JUL-83  0:00       512
 PROGRAMS        DIR        3   1-SEP-83
17:51  28-MAR-83  0:00      1536
 DATA            DIR        1   1-SEP-83
17:46  28-MAR-83  0:00       512
 EXTRAS          DIR        1   1-SEP-83
 0:00  28-MAR-83  0:00       512
 POSTAGE.RATES   BAS        8   1-SEP-83
 0:00  17-JUL-83  0:00      3273
 RENUMBER        BAS            1-SEP-83
 0:00  28-MAR-83  0:00

BLOCKS FREE:  107      BLOCKS USED:  173
    TOTAL BLOCKS:  280
```

Figure 3-2—Continued. CAT and CATALOG: 80-column CATALOG

```
/EXAMPLES
 NAME            TYPE  BLOCKS  MODIFIED         CREATED          ENDFILE SUBTYPE

 PRODOS           SYS      29   1-SEP-83  0:00  27-JUL-83 17:16    14336
 BASIC.SYSTEM     SYS      21   1-SEP-83  0:00   2-AUG-83 21:04    10240
 STARTUP          BAS       7   1-SEP-83 11:41  22-JUL-83  0:00     2750
 HELP             BIN       1   1-SEP-83 11:27  22-JUL-83 11:27       54
 HELPSCREENS      TXT      59   1-SEP-83  0:00  22-JUL-83  0:00 12358656
 DIRECTORY        DIR       1   1-SEP-83  0:00  28-MAR-83  0:00      512
 PRACTICE         DIR       1   1-SEP-83  0:00  18-JUL-83  0:00      512
 PROGRAMS         DIR       3   1-SEP-83 17:51  28-MAR-83  0:00     1536
 DATA             DIR       1   1-SEP-83 17:46  28-MAR-83  0:00      512
 EXTRAS           DIR       1   1-SEP-83 17:53  28-MAR-83  0:00      512
 POSTAGE.RATES    BAS       8   1-SEP-83  0:00  17-JUL-83  0:00     3273
 RENUMBER         BAS           1-SEP-83  0:00  28-MAR-83  0:00

BLOCKS FREE:  107      BLOCKS USED:  173      TOTAL BLOCKS:  280
```

For example, to see the files of jokes in the directory /JOKES/BAD, you can use the command in either of the following forms:

CATALOG /JOKES/BAD
CAT /JOKES/BAD

The Options

If you give the CAT or CATALOG command without any options, the names and characteristics of all the files in the prefix directory are displayed. If the prefix is empty, all the files in the volume directory indicated by the last used values of [,S#] and [,D#] are displayed.

[pn] The filename used in the command must indicate a directory file or you see the `FILE TYPE MISMATCH` error message.

[,S#] If you use the slot and drive options without a filename, ProDOS displays a list of the files in the

[,D#] volume directory of the disk in the specified drive.

For Example

To see a list of the files in the volume directory of the /EXAMPLES disk, type

```
CAT /EXAMPLES
```

and you get a list of the files in the volume named /EXAMPLES, *no matter which disk drive /EXAMPLES is in.* The display is similar to the one shown in Figure 3-3.

Figure 3-3. A Catalog of the /EXAMPLES Disk

```
/EXAMPLES
 NAME            TYPE   BLOCKS   MODIFIED

 PRODOS           SYS       29   1-SEP-83
 BASIC.SYSTEM     SYS       21   1-SEP-83
 STARTUP          BAS        7   1-SEP-83
 HELP             BIN        1   1-SEP-83
 HELPSCREENS      TXT       59   1-SEP-83
 DIRECTORY        DIR        1   1-SEP-83
 PRACTICE         DIR        1   1-SEP-83
 PROGRAMS         DIR        3   1-SEP-83
 DATA             DIR        1   1-SEP-83
 EXTRAS           DIR        1   1-SEP-83
 POSTAGE.RATES    BAS        8   1-SEP-83
 RENUMBER         BAS

BLOCKS FREE:  107      BLOCKS USED:  173
```

If your computer is an Apple IIc, put your /EXAMPLES disk in the built-in drive. Other Apple II's—put your /EXAMPLES disk in slot 6, drive 1. Now, type

```
CAT,S6,D1
```

and you get a list of the the files in the volume directory of *whatever disk is in the built-in drive, or the drive connected to slot 6, drive 1.* In this case, it's /EXAMPLES again.

Now look at the files in another directory. Notice that one of the files displayed on the screen is named PROGRAMS. Type

```
CAT /EXAMPLES/PROGRAMS
```

and a new list of files appears on the screen. These are the files stored on the /EXAMPLES disk in the directory named PROGRAMS.

What It All Means

By now you must be a little curious about the meaning of the headings that appear on the screen each time you use the CATALOG command. Type

```
CATALOG /EXAMPLES/DIRECTORY
```

and, if you have an 80-column display, you see a catalog resembling the one in Figure 3-4.

Figure 3-4. A Catalog of /EXAMPLES/DIRECTORY

```
DIRECTORY
 NAME             TYPE  BLOCKS  MODIFIED          CREATED           ENDFILE SUBTYPE

 DIRECTORY         DIR       1   1-SEP-83 11:31   18-JUL-83 15:09       512
 TEXT              TXT       1   1-SEP-83 11:31   18-JUL-83 10:09         0 R-     0
 APPLESOFT.PROG    BAS       1   1-SEP-83 11:31   18-JUL-83 15:10         0
 APPLESOFT.VARS    VAR       1   1-SEP-83 11:32   18-JUL-83 15:10         0
 SYSTEM.PROGRAM    SYS       1  <NO DATE>         18-JUL-83 15:10         0
 BINARY            BIN       1  <NO DATE>         18-JUL-83 15:10         0 A=$2000
 RELOC.CODE        REL       1   1-SEP-83 11:32   18-JUL-83 15:10         0
 USER.DEFINED.1    $F1       1   1-SEP-83 11:33   18-JUL-83 15:11         0

BLOCKS FREE:  107      BLOCKS USED:  173      TOTAL BLOCKS:  280
```

This is the 80-column version of this command. If you have a 40-column display, you see the same information, but each entry in the list takes up two lines on the screen.

The Directory's Name

The filename of the directory whose files you see in Figure 3-4 appears in the upper-left corner of the catalog. If it is a volume directory, it is preceded by a slash; otherwise it is not.

NAME (Filenames)

The name of each file in the directory is listed beneath the leftmost heading, the one labeled **NAME**. If the named file is locked, its filename is preceded by an asterisk.

TYPE (File Types)

The abbreviations beneath the heading labeled **TYPE** tell you the type of each file in that directory. The file type that corresponds to each abbreviation is shown in Table 3-1.

Table 3-1. The File Type Abbreviations

Abbreviation	File Type
DIR	Directory
TXT	Text
BAS	Applesoft Program
VAR	Applesoft Variables
BIN	Binary
REL	Relocatable Code
$F# †	User Defined
SYS	ProDOS System File
SYS	ProDOS System Program

† # is an integer from 1 to 8.

Notice that a file of each type is represented in the /EXAMPLES/DIRECTORY directory. With ProDOS started up and the /EXAMPLES disk in drive 1, type

```
HELP FILE
```

to see this list of file type abbreviations.

The uses for each file type are explained later in the manual.

BLOCKS (File Sizes)

A block is a 512-byte unit of disk space.

BLOCKS lists the number of blocks of disk space that each file uses.

For directory files, this column lists the number of blocks used by the directory file, but not the blocks used by the files in the directory.

When you catalog a volume directory, the block use of the entire disk is displayed at the bottom of the screen.

MODIFIED and CREATED (File Dates)

These columns contain the dates and times at which you created and last modified your files. The first half of the MODIFIED column is displayed by the CAT command; both columns are displayed by the CATALOG command. These dates and times are correct only if

- you have used the TIME program, described in Appendix D, or
- you have a clock/calendar card in your computer, set up as explained in Chapter 9.

ENDFILE (Maximum File Sizes)

Chapter 7 describes random-access text files and explains how this might happen.

ENDFILE lists the number of bytes that each file will use if all the disk space allotted to that file is filled (sometimes it is not).

SUBTYPE (File Properties)

SUBTYPE lists important properties of some types of files. A single letter precedes each number in this column. The letters used are shown in Table 3-2.

Table 3-2. The Subtype Column

Letter	Meaning
A	Load Address: This is the memory address from which a binary file (BIN) was saved. The address is given in hexadecimal.
R	Record Length: This is the size (in bytes) of each element in a text file (TXT). The length is in decimal.

For example, binary files are usually placed in the same part of memory each time they are used. For a binary file the property in the **SUBTYPE** column starts with an A (for load Address) followed by the memory address at which that binary file was last placed. This type of attribute is discussed in Chapter 4, "BASIC Programs in Files."

The other type of property begins with an R (for Record length). It specifies the size of each element of the file. This attribute is explained in Chapter 6, "Text In Files."

The Bottom Line

When you display the catalog of a directory, the bottom line of the display describes how space on the volume is used.

BLOCKS FREE is the number of unused blocks on the disk, **BLOCKS USED** is the number of full blocks on the disk, and **TOTAL BLOCKS** is the maximum number of blocks of information that the disk can hold.

The PREFIX Command

If you are going to be referring to several files in a single directory, the PREFIX command can save you some time. Use the PREFIX command to set the prefix to the name of the directory the files are in; you can then refer to the files by filename alone.

To assign a new value to the prefix, or to see the current value of the prefix, use the command:

/PREFIX [pn] [,S#] [,D#]

For example, if the prefix is set to /EXAMPLES/, and you want to use the file named /MAMMALS/RODENTS/BEAVER you must refer to it by its full pathname. If you use the command

PREFIX /MAMMALS/RODENTS/

you can then refer to the file simply as BEAVER, and to the other files in the directory as MOUSE, SQUIRREL, RAT, and so on.

When you start up /EXAMPLES or any other ProDOS disk, the prefix is left empty, and the slot and drive defaults are set to indicate the drive containing that disk. Whenever the prefix is empty, ProDOS looks for files on the disk in the indicated slot and drive.

The Options

If you use the PREFIX command without any options, the current value of the prefix is displayed on the screen.

[pn] — pn must be the pathname or partial pathname of a directory file. Unlike a normal pathname or partial pathname, it may end with a slash. If it is not valid, you will get a `FILE TYPE MISMATCH` or `FILE NOT FOUND` error.

If you use a slash instead of a pathname, the value of the prefix is left empty.

[,S#]
[,D#] — If you do not specify a filename, but you do use the slot and drive options, the volume name of the indicated disk is assigned to the prefix. If you have a controller for a single disk drive in a slot, refer to it as drive 1 of that slot.

For Example

With ProDOS started up, and the ProDOS disk in drive 1, set the prefix to indicate the /EXAMPLES volume directory using the command

```
PREFIX /EXAMPLES
```

and then make sure the command worked right using the command

```
PREFIX
```

Notice that the value of the prefix is printed out as

```
/EXAMPLES/
```

If the prefix value that you assign doesn't end in a slash, ProDOS automatically adds one. It does this so, when the prefix is attached to a filename or partial pathname, a proper pathname is formed.

Verify that you can enter the prefix with a trailing slash by typing

```
PREFIX /EXAMPLES/
```

and checking the result using the command

```
PREFIX
```

If you want to set the prefix to the name of the volume directory of one of your disks, but you can't quite remember the disk's name, use the slot and drive options to indicate the disk's location. For example

```
PREFIX, S6, D1
```

sets the prefix to the value /EXAMPLES/. Try it. If your disk has a long name, this form may be shorter than typing in the entire volume name.

If you want to look at or use some of the programs that are supplied with your /EXAMPLES disk, type the command

```
PREFIX /EXAMPLES/PROGRAMS/
```

You can now see a list of the programs in this directory by typing

```
CAT
```

and you can refer to any one of them by filename alone, as in

```
RUN WHIZBOOM
```

When You Use PREFIX in a Program

Refer to Chapter 5, "Programming With ProDOS," for details on reading the prefix.

When you use the PREFIX command with no options from within a program, the value of the prefix is not displayed; it is set up so the next INPUT statement in the program reads it.

The CREATE Command

The primary purpose of the CREATE command is to create directory files within which you can place other files. Although you can use this command to create files of all types, most of the other file types are created automatically by other ProDOS commands.

A volume directory file can store the names and locations of up to 51 files. It isn't really necessary to create more directory files unless your disk will contain more than 51 files. However, a well planned set of directories can make your files much easier to find and use.

Be Prepared

Create your directory files before you have files to place within them. It is much easier to place new files in a directory than to move existing files from one directory to another. If you must do this, use the file copy option of the ProDOS Filer.

You create a file by using the command

CREATE pn [,Ttype] [,S#] [,D#]

Notice that the CREATE command uses a new option, Ttype, that determines the type of file to be created. If you do not use this option, a directory file is created. To create a file of any other type, you must use this option.

For example, you can create a directory file named /BUDGET/CHILDREN (on a disk named /BUDGET) by using the command

```
CREATE /BUDGET/CHILDREN
```

Directory File Size

The number of files that fit into a directory other than a volume directory is limited only by the amount of space on the disk. The size of a directory file is determined by the number of files it contains.

The first block of disk space used by a directory can hold up to 12 files. Each subsequent block used by a directory can hold up to 13 files. Thus a directory with 27 files in it is three blocks long (12 files in first block, 13 in second block, 2 in third block).

The Options

pn	pn is the pathname or the partial pathname of the file to be created. The file must not exist.
[,Ttype]	type is a three-letter abbreviation that determines the type of file to be created. The abbreviations of the various file types are given in Table 3-3. You can see these abbreviations by using the HELP FILE command.

Chapter 2 discusses the directory file type; the other types are explained as you need to know about them.

Table 3-3. The File Type Abbreviations

Abbreviation	File Type
DIR	Directory
TXT	Text
BAS	Applesoft Program
VAR	Applesoft Variables
BIN	Binary
REL	Relocatable Code
$F# †	User Defined
SYS	ProDOS System File
SYS	ProDOS System Program

† # is an integer from 1 to 8.

[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

For Example

With ProDOS running and the /EXAMPLES disk in drive 1, set the prefix to /EXAMPLES/PRACTICE/ with the command

PREFIX /EXAMPLES/PRACTICE/

We want to create a directory named /EXAMPLES/PRACTICE/NEWDIR. Type the command

```
CREATE NEWDIR
```

and listen to the /EXAMPLES disk whirring away. Now type

```
CAT
```

to see that the new directory exists. Notice that the catalog of files shows that **NEWDIR** uses up 1 block of disk space.

The RENAME Command

To change the name of a file, use the RENAME command:

RENAME pn1,pn2 [,S#] [,D#]

This command changes the name of a file from the name indicated by pn1 to the name indicated by pn2. The new name must be in the same directory as the old name. Thus you can use the command

```
RENAME
/RECIPES/TEST/FUDGE,/RECIPES/TEST/CANDY
```

to change the name of a test recipe from FUDGE to CANDY, but you *cannot* use the command

```
RENAME
/RECIPES/TEST/CANDY,/RECIPES/EDIBLE/CANDY
```

to move your CANDY recipe from the TEST directory to the EDIBLE directory. To move a file from one directory to another, use the ProDOS Filer.

You cannot rename a file that is locked. Refer to the section "The LOCK Command" in this chapter for further details.

The Options

pn1,pn2	When you give a file (pn1) a new name (pn2), the new name must be unique. If it already exists, you get the `DUPLICATE FILE` name error message. If pn1 does not exist, you get the `FILE NOT FOUND` error message. If the two pathnames, pn1 and pn2, do not indicate files in the same directory, you get a `SYNTAX ERROR`.
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

For Example

With the /EXAMPLES disk in drive 1, set the prefix to the name of the directory containing practice files by typing

```
PREFIX /EXAMPLES/PRACTICE/
```

Now look at the files in the directory /EXAMPLES/PRACTICE by typing

```
CAT
```

The list of files displayed on the screen includes the files RENAME.ME.1, RENAME.ME.2, and RENAME.ME.3. To change the name RENAME.ME.1 to RENAME.ME.4, type

```
RENAME RENAME.ME.1, RENAME.ME.4
```

and ProDOS swiftly and silently changes the file's name. Now type

```
CAT
```

to verify that the name was indeed changed. Now rename the file LOCKED.UP.1 with the command

```
RENAME LOCKED.UP.1,INMATE
```

Whoops! This file is locked; its name cannot be changed until you unlock it. If you have valuable files that you don't want to accidentally alter or rename, lock them. The sections on the LOCK and UNLOCK commands in this chapter describe this method of file protection.

The DELETE Command

To remove a file from a disk, use the command

DELETE pn [,S#] [,D#]

For example, you can remove the file /RESUME/DRAFT12 from its disk with the command

```
DELETE /RESUME/DRAFT12
```

Be Careful

The file indicated by pn must be unlocked, and if it is a directory file, it must be empty. After you delete a file there is no way to get it back again.

The Options

pn	A pathname or partial pathname, pn, must be included in the command; the indicated file must exist or you get a `FILE NOT FOUND` message.
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

For Example

With ProDOS started up, and the /EXAMPLES disk in drive 1, set the prefix to /EXAMPLES/PRACTICE/ with the command

```
PREFIX /EXAMPLES/PRACTICE/
```

To see the files in the PRACTICE directory, type the command

```
CAT
```

Included in the listed files are the files DELETE.ME.1, DELETE.ME.2, and DELETE.ME.3.

Now delete the file /EXAMPLES/PRACTICE/DELETE.ME.1 using the command

```
DELETE DELETE.ME.1
```

Try deleting the other DELETE.ME files using a full pathname, and by setting the prefix to something else, /EXAMPLES/ for example, and using a partial pathname.

Use the CAT command to verify that the deleted files are no longer on the disk.

The LOCK Command

At times you will want to protect individual files from being accidentally renamed, deleted, or altered. You can do this using

LOCK pn [,S#] [,D#]

For example, you can lock the file /INVENTORY/NOTE.PADS by typing

```
LOCK /INVENTORY/NOTE.PADS
```

While a file is locked, it cannot be renamed, deleted, or altered. Any attempt to change a locked file causes the `FILE LOCKED` error message to be displayed. To alter a locked file, you must first unlock it with the UNLOCK command.

When you catalog the files in a directory, the locked files are marked by asterisks to the left of their filenames.

You cannot lock a volume directory file. You can, however, protect an entire flexible disk by covering its write-enable notch.

The Options

pn	pn is the pathname or the partial pathname of the file to be locked. You cannot lock a volume directory.
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

For Example

With ProDOS started up and the /EXAMPLES disk in drive 1, display a list of the files in the PRACTICE file using the command

```
CAT /EXAMPLES/PRACTICE
```

Notice that the files LOCKED.UP.1 and LOCKED.UP.2 both have asterisks by their file types, indicating that they are locked. First set the prefix to /EXAMPLES/PRACTICE using the command

```
PREFIX /EXAMPLES/PRACTICE
```

Next lock the file LOCK.ME.1 in the prefix directory with the command

```
LOCK LOCK.ME.1
```

Use the CAT command to verify that the file is now locked.

The UNLOCK Command

Before you can delete, rename, or otherwise change a locked file, you must UNLOCK it using the command

UNLOCK pn [,S#] [,D#]

For example, to unlock the file /INVENTORY/NOTE.PADS so that you can update the information it contains, use the command

```
UNLOCK /INVENTORY/NOTE.PADS
```

You can UNLOCK any file except a volume directory file. Volume directory files cannot be locked.

The Options

pn	pn is the pathname or the partial pathname of the file to be unlocked.
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

For Example

With ProDOS started up and the /EXAMPLES disk in drive 1, display the list of the files in the PRACTICE file by using the command

```
CAT /EXAMPLES/PRACTICE
```

Notice that the files LOCKED.UP.1 and LOCKED.UP.2 both have asterisks by their file types, indicating that they are locked. First set the prefix to /EXAMPLES/PRACTICE/ using the command

```
PREFIX /EXAMPLES/PRACTICE
```

Next unlock the file LOCKED.UP.1 in the prefix directory with the command

```
UNLOCK LOCKED.UP.1
```

Use the CAT command to verify that the file is now unlocked.

The VERIFY Command

Use this command to verify that a particular file exists. You might do this, for example, before creating a file—to be sure that another file with that name doesn't already exist.

VERIFY pn [,S#] [,D#]

If the file already exists, no message appears. If no file with the specified name already exists, you'll see the message **PATH NOT FOUND**.

The Options

pn	pn is the pathname or the partial pathname of the file that you want to verify exists.
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

Chapter 4 BASIC Programs in Files

About This Chapter

This chapter describes the ProDOS commands that let you use the BASIC programs on your disks. If all you want to do is run programs that are already on your disks, pay special attention to the —(DASH) command. This command moves any type of program from a file on a disk into memory, and then starts it running.

Use the commands in this chapter

- to bring a program into memory and run it
- to bring a program into memory without running it
- to store on a disk the program that is currently in memory.

These commands are most useful when you are writing programs or modifying programs that already exist.

BASIC Program Files

When you start up /EXAMPLES, it displays the message

```
APPLESOFT IN ROM
```

telling you that the Applesoft language is in read-only memory.

Did You Remember? Although computers in the Apple II family can use two dialects of the BASIC language—Applesoft and Integer BASIC—ProDOS supports only Applesoft. To use ProDOS, your Apple II must have Applesoft BASIC in read-only memory (ROM), and at least 64K of random-access memory (RAM).

This Chapter's Commands

This chapter's commands are summarized below. Each causes the transfer of a program between memory and a disk file; the direction of the transfer and the type of file transferred are determined by the command you use.

—(DASH): Run any type of program

The EXEC command is described in Chapter 8.

This command is a convenient short form of the RUN, BRUN, and EXEC commands. It transfers a BASIC, binary, EXEC, or system program from a disk file into memory and executes it. You don't have to know a program's file type to use the DASH command.

RUN: Run a BASIC program from a file

Binary program execution: see also the DASH command.

Use this command to copy a BASIC program, file type BAS, from a disk file into memory to be executed automatically.

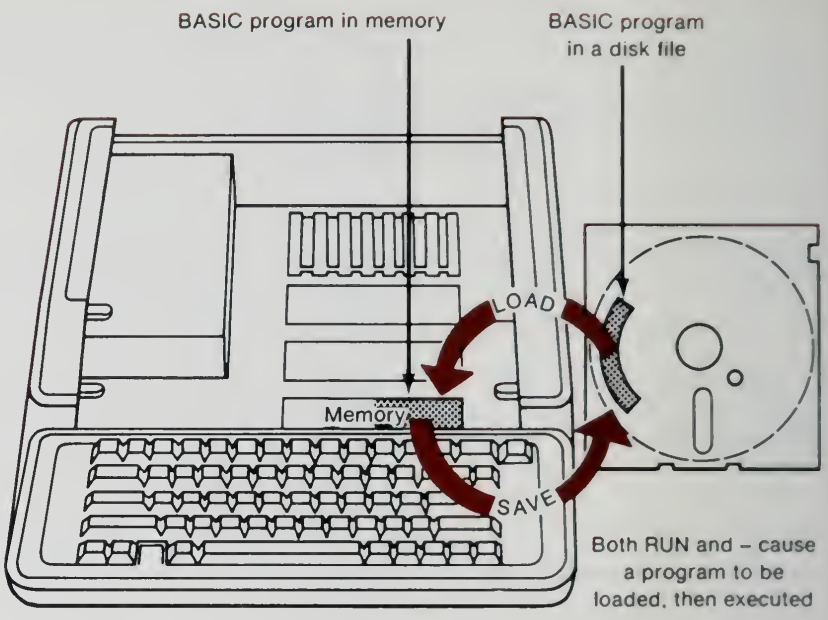
LOAD: Get a BASIC program from a file

Use this command to copy a BASIC program, file type BAS, from a disk file into memory. Once the program is in memory, you can run it, modify it, or save it in a disk file.

SAVE: Save a BASIC program in a file

Use this command to save the BASIC program that is currently in memory as a BASIC disk file, file type BAS.

Figure 4-1. BASIC in Files



The –(DASH) Command

One of the more useful features of ProDOS is the –(DASH) command. With this command you can bring into memory and run: a BASIC program, a machine-language program, an EXEC program, or a system program.

EXEC program: see Chapter 8.

To learn more about the way programs run, read the sections "The RUN Command" in this chapter and "The BRUN Command" in Chapter 9.

To run a program of any one of these types, use the command

–pn [,S#] [,D#]

▲Warning

When you run a system program, everything else in memory is destroyed. If you are writing a BASIC program, be sure to save it before running a system program.

The Options

pn	pn is the pathname or partial pathname of the file containing the program you want to run. The file must be of type BAS, BIN, TXT, or SYS. If the file is a binary file, it is loaded to the address from which it was last saved. All other file types cause the `FILE TYPE MISMATCH` error.
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

For Example

With ProDOS running, set the prefix to indicate the PROGRAMS directory with the command

```
PREFIX /EXAMPLES/PROGRAMS/
```

Look at the programs it contains by typing

```
CAT
```

Now try running a binary program (type BIN), a BASIC program (type BAS), or an EXEC program (type TXT) by typing a dash (–) followed by the filename of the program you want to run.

The RUN Command

To run an Applesoft program that is stored on a disk, use either of the commands

RUN pn [,@#] [,S#] [,D#]

The — (DASH) command is described earlier in this chapter.

—pn [,S#] [,D#]

When ProDOS sees the RUN command, it finds the file indicated by pn, brings it into memory, and then runs it (beginning with line @# if you use that option). For example, to run the BASIC program named CHECKBOOK on a disk named /ACCOUNTS, use the command

```
RUN /ACCOUNTS/CHECKBOOK
```

When a program ends, you can run it again by using the BASIC command

Notice that in the general form of the RUN command, the filename is not optional. When you type in a command, ProDOS checks to see if it is a ProDOS command. If it isn't, ProDOS gives the command to BASIC. In this case, RUN without a filename is not a valid ProDOS command, the command is passed to BASIC, and BASIC runs the program in memory.

The Options

pn	pn indicates the file to be run. The file to be run must be of type BAS (Applesoft BASIC).
[#]	If you use this option, the program in memory begins running at the line number specified by #. If you omit this option, excution begins at the lowest numbered line in the program.
[,@#]	Use this option to start a program *on disk* (not yet in memory) running at a specified line.
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

For Example

With ProDOS started up, and the /EXAMPLES disk in drive 1, set the prefix to indicate the /EXAMPLES/PROGRAMS volume directory using the command

```
PREFIX /EXAMPLES/PROGRAMS/
```

Look at the files in the PROGRAMS directory with the command

```
CAT
```

Do you see a program named **TWO.LINER** ? Run the program by typing

```
RUN TWO.LINER
```

and the screen displays the words

```
I'M LINE 10.  I'M LINE 20.
```

But if you run part of the program using the command

```
RUN TWO.LINER 20,
```

you see just

```
I'M LINE 20.
```

If you examine the program by typing

```
LIST
```

you'll discover that line 20 contains that sentence.

The LOAD Command

To transfer a BASIC program from a file to memory, use the command

LOAD pn [,S#] [,D#]

This command is useful if you want to examine or modify a program. It is not necessary to load a program before you run it; the RUN command automatically loads a program into memory.

▲Warning

When a new program is loaded into memory, all records of the previous program are erased from the Apple II's memory. If you want to save the variables that were set by a previous program, use the STORE command.

The STORE command is described in Chapter 5.

When you LOAD the contents of a file into memory, the file on the disk remains unchanged.

Once a program is loaded into memory, you can run it by simply typing **RUN**.

The Options

pn	pn must indicate an Applesoft program file (type BAS).
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

For Example

With ProDOS started up, and the /EXAMPLES disk in drive 1, set the prefix to indicate the /EXAMPLES/PROGRAMS volume directory using the command

```
PREFIX /EXAMPLES/PROGRAMS/
```

Type **NEW** to remove any BASIC program from memory, then type **LIST** just to prove that there is no BASIC program in memory. Now, bring a new program into memory with the command

```
LOAD WHIZBOOM
```

You hear the disk drive working away, then the BASIC prompt returns. To see the program that was just loaded, type

```
LIST
```

It is important to realize that when you load a file, a copy of the file is transferred from the disk to memory, but the original copy of the file remains unchanged on the disk. For example, even though you have already loaded WHIZBOOM, type the command

```
CAT
```

and you see that the file is still on the disk. To place the BASIC program that is currently in memory into a file, use the SAVE command.

The SAVE Command

To transfer the BASIC program that is currently in memory to a file on a disk, use the command

SAVE pn [,S#] [,D#]

The file is saved as an Applesoft file (type BAS).

For example, to save the current program in a file named /BEDTIME.STORIES/A.FRIENDLY.OGRE, use the command

```
SAVE /BEDTIME.STORIES/A.FRIENDLY.OGRE
```

To save a program to a filename that already exists, the file must not be locked, and it must be of the same type as the program (type BAS).

▲Warning If you are working with several different program files at once, and there is a chance you will save a program to the wrong file, LOCK each file after you save to it, and then UNLOCK it before you save to it again. This is the best way to protect valuable BASIC programs.

By the Way: You can use the SAVE command to rearrange program files. If you use the LOAD command to bring a program into memory, you can then use the SAVE command to save it on a different disk (you would then have two copies of the same file). Remember that when you load a BASIC program from a file, the file itself does not change.

The Options

pn	pn is the pathname or the partial pathname of the file in which you want to save the current program. If pn already exists, it must be unlocked.
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

For Example

With ProDOS started up, and the /EXAMPLES disk in drive 1, set the prefix to /EXAMPLES/PROGRAMS with the command

```
PREFIX /EXAMPLES/PROGRAMS/
```

Now load a very short program into memory with the command

```
LOAD VERY.SHORT
```

To see how short this program is, type the command

```
LIST
```

and you see before you a single line of program which reads

```
10  PRINT "A VERY SHORT PROGRAM"
```

Even if you aren't a programmer, you might guess that this program does one and only one thing: it prints the words **A VERY SHORT PROGRAM** onto the screen. To see it do its very short thing, type

```
RUN
```

Remember that the RUN command without a filename is not a ProDOS command; it is a BASIC command that causes the program that is currently in memory to be run.

Now for the fun part. Save the program to a different disk file, say /EXAMPLES/PROGRAMS/ONE.LINER, with the command

```
SAVE ONE.LINER
```

You can see that the program has found its new home by typing

```
CAT
```

Notice that the first copy of the program, VERY.SHORT, is still on the disk. Notice also that a program, even after a SAVE, remains in memory. Use **RUN** or **LIST** to see that the program is still there.

If you have an Apple IIc, or an Apple IIe with an Extended 80-Column Text Card, you can also save this program to a file on the volume named /RAM. Type the command

```
SAVE /RAM/ONE.LINER
```

You can see that this version of the program has been saved by using the command

```
CAT /RAM
```

Notice how much faster /RAM is than a normal disk volume.

If you're working with two different BASIC programs, and you need to switch often from one to the other, you can save them all to /RAM. Then, running them from /RAM is much faster than from disk, especially if they're stored on two different disk volumes.

▲Warning

Another use for /RAM is when you're developing a large BASIC program and you wish to save intermediate versions of it frequently. But remember that all the contents of the /RAM volume will disappear in the event of a power failure!

Chapter 5

Programming With ProDOS

About This Chapter

This chapter contains an overview of programming with ProDOS. It describes

- the startup process and how to make a startup disk
- how to use ProDOS commands from within a program
- how to read the prefix from within a program
- how a program can handle errors
- how your programs can communicate with devices in slots.

You should have some experience with Applesoft BASIC. If you did all the examples in the *Applesoft Tutorial*, you know enough. If you didn't, you should work your way through it before continuing with this manual.

If you know Integer BASIC, but not Applesoft, read the appendix in the *Applesoft BASIC Programmer's Reference Manual* that summarizes the differences between the two.

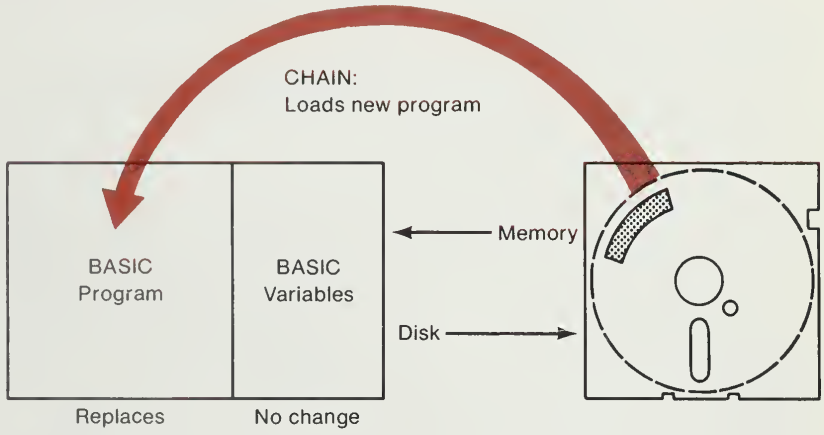
This Chapter's Commands

The commands described in this chapter are summarized below. This is not meant to be a complete description of each command; it simply gives you an idea of the use of each command and of the way the commands are interrelated.

CHAIN: Run a program, but save the variables

Use this command to run a new program. Unlike RUN, it does not cause the variables that are already in memory to be thrown away. See Figure 5-1.

Figure 5-1. CHAIN



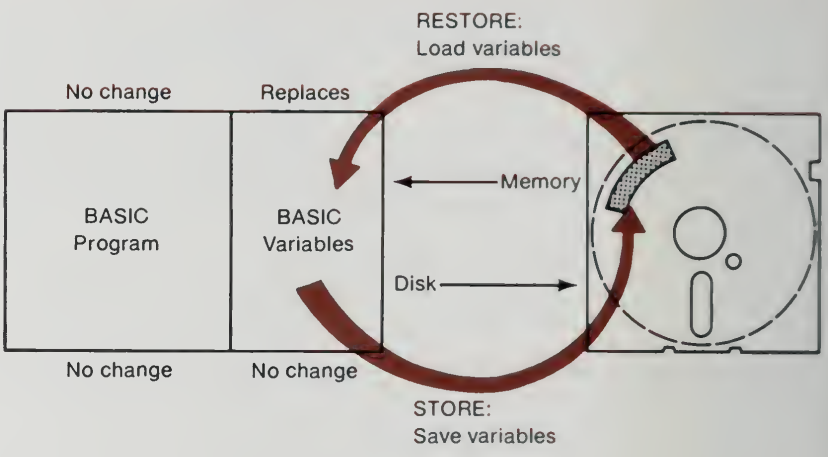
STORE: Save the variables in memory to a file

Use this command to save in a file all the BASIC variables that are currently in memory. You can retrieve these variables by using the RESTORE command.

RESTORE: Get BASIC variables from a file

Use this command to discard the variables that are currently in memory, and to bring in new variables from a file.

Figure 5-2. STORE and RESTORE



PR#: Send output to a slot

Chapter 9 explains how to use PR# to send output to a program.

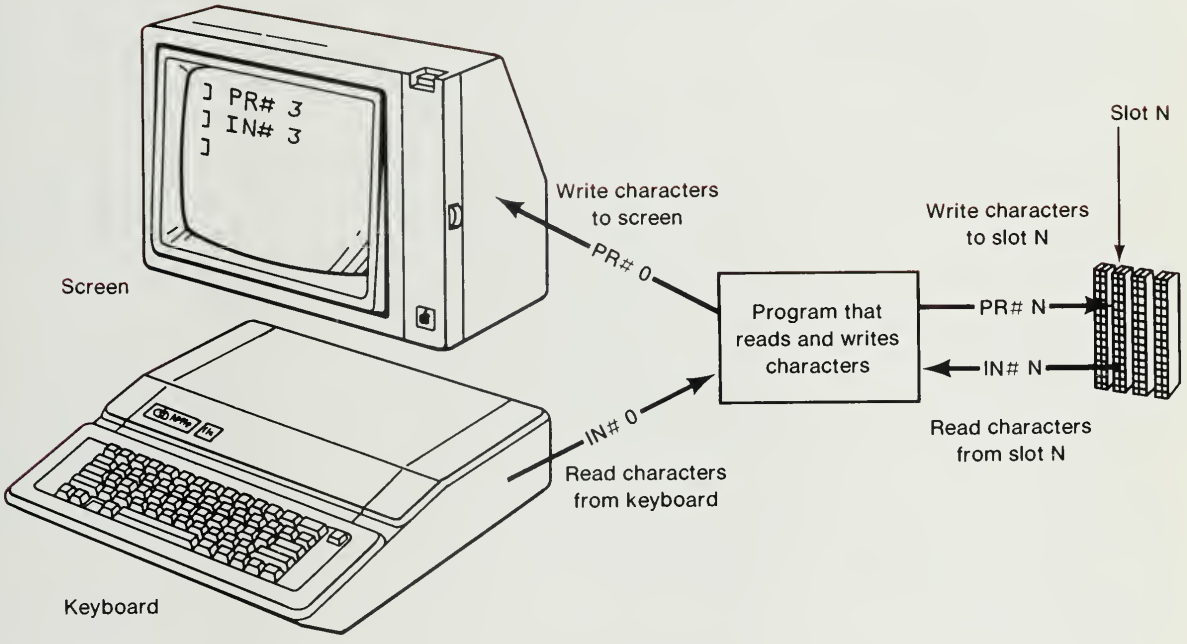
Use this command to cause the characters that are normally printed on the screen to be sent to a device, such as a printer, in a slot.

IN#: Get input from a slot

Chapter 9 explains how to use IN# to get input from a program.

Use this command to cause characters to be read from a device, such as an external terminal, in a slot instead of from the keyboard.

Figure 5-3. PR# and IN#



FRE

Use this command to access ProDOS' housekeeping routines.

BYE

Use this command to leave Applesoft BASIC and start another application program without going through the startup procedure.

What Is a Startup Disk?

Every Apple II system has a particular disk drive known as the system's startup drive. In an Apple IIc, this is the built-in drive.

In other computers in the Apple II family, the startup drive is the one connected to drive 1 of the disk controller card in the highest numbered slot that contains a disk controller. This chapter assumes that your startup drive is connected to slot 6.

When you place a disk in the startup drive and turn on your Apple II, some disks cause the system to start up, other disks just spin. The disks that cause the system to start up are known as **startup disks**. A startup disk contains all the crucial information that the Apple II needs to bring a program from a disk into the Apple II's memory and start a program running.

A ProDOS startup disk contains all the information needed to bring ProDOS into memory and start it running.

To make any ProDOS-formatted disk into a ProDOS startup disk, you must copy the files containing the ProDOS program onto that disk. When you turn on the Apple II (or type `PR# 6` from BASIC) with a ProDOS startup disk in the startup drive, the ProDOS program is automatically transferred from disk to memory and run.

Once you have made other startup disks, you will no longer need to use the /EXAMPLES disk each time you use ProDOS. All the vital parts of ProDOS are stored on each startup disk. In addition, by copying the files HELP and HELPSCREENS onto the startup disk, you can use the help feature with other startup disks.

You can designate any program to be run automatically when a ProDOS disk is started. You do so by putting the program in a file named STARTUP. This file, which may contain a binary, BASIC, or EXEC program, can contain a program that places a greeting message on the screen, a favorite game, budgeting program, or other program of your choice.

Examples of STARTUP programs are given later in this chapter.

The Anatomy of a Startup Disk

A ProDOS startup disk has three characteristics:

- It is formatted, either by the *ProDOS User's Disk* or the *System Utilities* disk.
- It has the file PRODOS in its volume directory.
- It has a file named BASIC.SYSTEM in its volume directory.

To make a ProDOS startup disk:

1. Use either the *ProDOS User's Disk* or the *System Utilities* disk to format a disk.
2. Use either the *ProDOS User's Disk* or the *System Utilities* disk to copy the files PRODOS and BASIC.SYSTEM from the /EXAMPLES disk to the newly formatted disk.
3. If you want a specific program to be run when you start up that disk, place it on the startup disk and name it STARTUP.
4. If you wish to use the help commands after you start up from that disk, copy the files HELP and HELPSCREENS from the /EXAMPLES disk to the new startup disk.

The Startup Process

When you start up your Apple II with a proper ProDOS startup disk, here is what happens:

1. The file named PRODOS, containing the most sophisticated parts of ProDOS, is transferred from the startup disk into memory and run.
2. ProDOS examines the peripheral connector slots (or IIc ports), tries to identify the type of device in each slot/port, and determines where there are disk drives that it may need to communicate with. If one of the devices is a clock/calendar card that follows the ProDOS protocol, ProDOS sets itself up to read from the clock/calendar.
3. It then loads the program in the file named BASIC.SYSTEM. This portion of ProDOS contains all the ProDOS commands. This part of the program is run.
4. ProDOS BASIC looks for a file named STARTUP on the startup disk. If it finds one, it runs it. If it does not find one, it displays the following message:

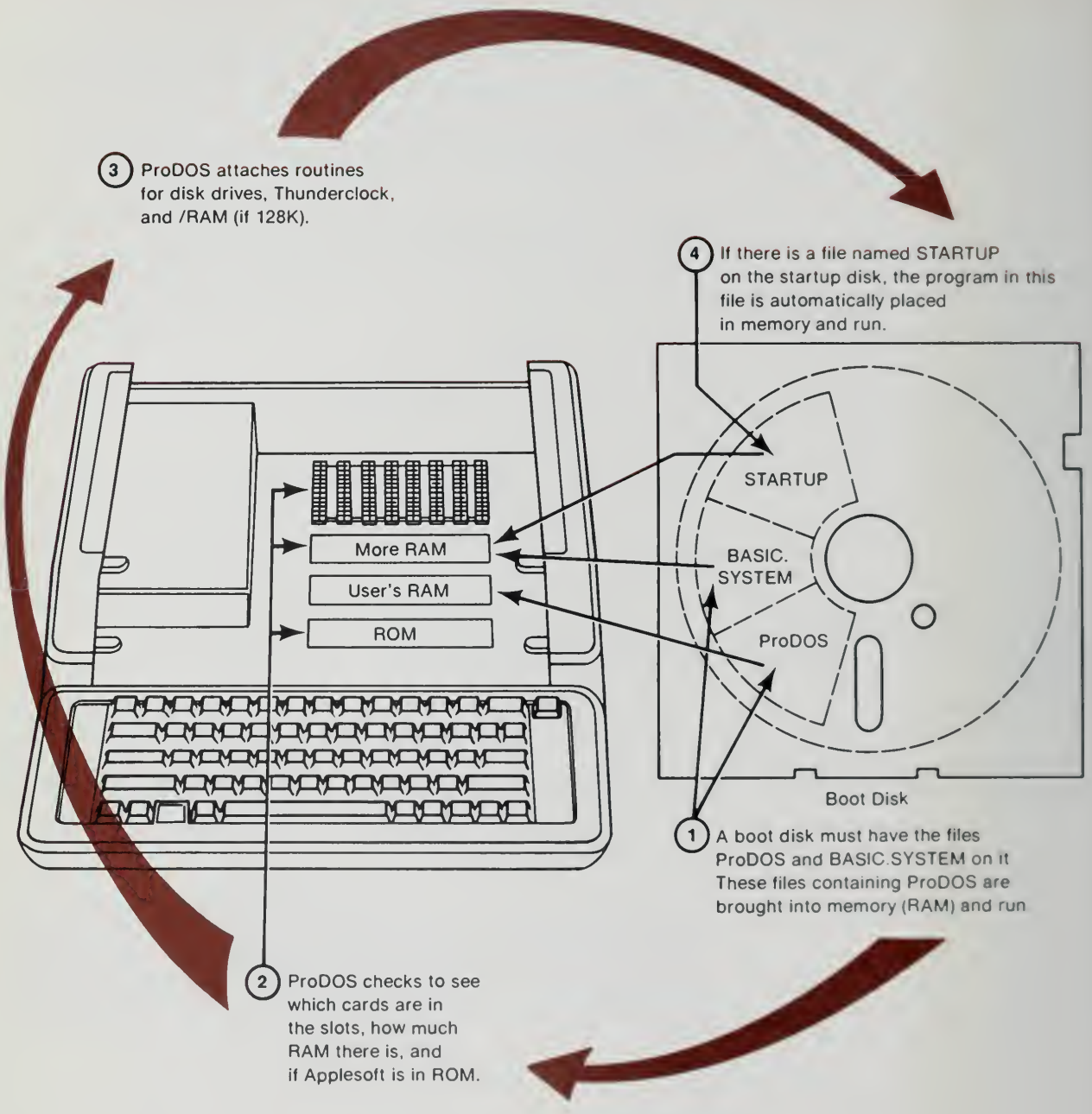
For more information on clock/calendar cards, see the *ProDOS Technical Reference Manual* or *Design Guidelines.*

```
PRODOS BASIC 1.1
COPYRIGHT APPLE, 1983-84

] ■
```

Figure 5-4 illustrates the steps in the startup process.

Figure 5-4. The Startup Process



Using ProDOS From Within a Program

Very often it's useful to be able to use a ProDOS command from within a BASIC program. For example, you can use ProDOS commands in a program

- to print out a disk's catalog from a STARTUP program
- to save your budget records in a file
- to save the condition of a game program for next time.

To use a ProDOS command from within a program, you must print a string consisting of a CONTROL-D as the first thing on a printed line, followed by the command. Using the CATALOG command as an example, here is one way to put CONTROL-D in a string:

```
10  PRINT "CATALOG"
```

There is a CONTROL-D between " and C.

Right after you type the first quotation mark, press CONTROL-D. Although you can't see it, it's there.

Here is another way to put CONTROL-D in a string:

```
10  PRINT CHR$ (4);"CATALOG"
```

CHR$ (4) returns CONTROL-D.

In this example, CHR$ is an Applesoft function that returns the character whose code is in parentheses. The number 4 is the Apple II's code for CONTROL-D. Note the semicolon after CHR$ (4). It is an optional separator between elements of a PRINT list, and you may leave it out if you want.

If you set the value of a string variable, say D$, to CONTROL-D at the beginning of a program, simply print **D$** before each ProDOS command in the program. This is the method used throughout the manual (see line 40 in the example below).

For Example

Here is a version of a STARTUP program that prints a message on the screen, and then prints out a list of the files on that disk. Type **NEW**, and then enter this program.

```
 5  REM APPLESOFT STARTUP
10  D$ = CHR$ (4)                    D$ contains (CONTROL)-(D).
20  PRINT "PRODOS TEST PROGRAMS"     Print a title.
30  PRINT " 26 NOVEMBER 1985"        And a date.
40  PRINT D$;"CATALOG"               Then list the volume directory.
50  END
```

Now type **RUN** to see how it works. If you want to preserve this program, put a ProDOS-formatted disk in drive 1, and type

```
SAVE STARTUP
```

Debugging Your Programs

If you are a former DOS user, you will be pleased to discover that the Applesoft commands TRACE and NOTRACE work with ProDOS.

Use the TRACE command to cause the line number of each BASIC program line that is executed to be printed on the screen. Use the NOTRACE command to stop the printing of line numbers.

Things to Watch Out For

This section describes two things you should watch out for while using ProDOS.

You Can't Copy Control Characters

When you use (→) to copy a BASIC statement, invisible control characters are not copied.

Some ProDOS Commands Work Only in Programs

Most ProDOS commands can be given from the keyboard as immediate commands or from within programs as deferred commands. Some can only be used from within programs. They are

APPEND
OPEN
POSITION
READ
WRITE

If you aren't sure if a ProDOS Command can be used from immediate mode, simply use the HELP command. The upper-right corner of each help screen tells how that command can be used (as IMM[editate] or DEF[erred] commands). For example, type `HELP WRITE` to display the WRITE help screen. You see that this command can only be used in a program.

Intercepting Messages to the Display Screen

You can use several ProDOS commands to cause ProDOS to send information to the screen. The CATALOG command produces a list of files, the PREFIX command without options displays the prefix, and, if you use any command incorrectly, an error message is sent to the screen.

Appendix D explains how to open and read directory files.

ProDOS provides a way for your program to read the value of the prefix or the number of an error message, these are explained below.

Reading the Prefix

When you give the PREFIX command without options from the keyboard, the current value of the prefix is displayed. When you use the PREFIX command from within a program, ProDOS does not display the value of the prefix. Instead, ProDOS places the value of the prefix so the next INPUT statement in your program will read it.

You might use this feature, for example, if your program changes the value of the prefix, and you want to restore the prefix to its former value before the program ends. Here is a program portion that does that.

```
10 D$ = CHR$ (4)                    CHR$(4) is CONTROL-D.
20  PRINT D$;"PREFIX"               Command to display prefix.
30  INPUT PR$                       Read the prefix into PREFIX$.

200  PRINT D$;"PREFIX ";PR$         Restore prefix to original value.
```

Notice the way this feature works: You give the prefix command without any options, and input the prefix into any string variable, in this case, PR$. When you want to restore the prefix, the old value is in the string.

Handling Errors in a Program

An error in an Applesoft program normally causes an error message to appear on the display screen. If your program uses the ONERR GOTO statement, Applesoft puts an error number in memory, and then, when there is an error, goes to the line specified in the ONERR GOTO statement.

ProDOS follows the same procedure. When it encounters an error, it places the error number in memory, and then tells Applesoft that it found an error. ProDOS uses error numbers not used by Applesoft. Table C-2 lists the numbers that are useful to a program trying to catch ProDOS or Applesoft errors. Table 5-1 lists memory locations for error handling.

Table 5-1. Memory Locations for Error Handling

Number	How to Read It
Error Number	PEEK (222)
Line Number	PEEK (218) + PEEK (219) * 256

For example, you cannot rename a file if it is locked. The error number for **FILE LOCKED** is 10. This small program lets you rename a file whether or not it is locked. Using ONERR GOTO, it detects the **FILE LOCKED** error, gives you a chance to unlock the file, then it tries to rename the file again. It also displays the error number and line number of any other error that might occur.

```
5  REM  ONERR.DEMO
10 D$ = CHR$ (4)                          D$ is (CONTROL)-(D).
20  ONERR GOTO 100                        Error handled at line 100.
30  INPUT "FILE TO RENAME? ";F$           Read a filename into F$.
40  INPUT "NEW NAME? ";N$                 Read the new name into N$.
50  PRINT D$;"RENAME ";F$;",";N$          Give the RENAME command.
60  END                                   No error, program ends.
```

Line 20 causes line 100 to be called if there is any error.

```
100  IF PEEK (222) <> 10 THEN 200         If it is not a FILE LOCKED error (10), go to
                                          line 200.
110  INPUT "FILE IS LOCKED. RENAME
     ANYWAY? (Y/N) ";Y$
120  IF Y$ <> "Y" AND Y$ <> "y"
     THEN END                             No, don't rename file.
130  PRINT D$;"UNLOCK ";F$                Yes, unlock the file, rename it, and then lock
                                          it again.
140  PRINT D$;"RENAME ";F$;",";N$
150  PRINT D$;"LOCK ";N$
160  END                                  All done.
```

Line 100 calls line 200 if there's an error other than **FILE LOCKED**.

```
200  PRINT "ERROR #";PEEK (222);" DETECTED"
210  PRINT "AT LINE ";PEEK (218) + PEEK
     (219) * 256
220  END
```

This program is saved with the name /EXAMPLES/PROGRAMS/ONERR.DEMO. Try it on a few of your locked files.

Its operation is quite simple. Line 30 reads the name of a file to be renamed into F$. In line 40 it reads the new name into N$. In line 50 the RENAME command is given. If there is no error, the file is renamed, and the program ends. If there is an error, the ONERR statement in line 20 says that the program should look to line 100 for the part of the program that treats the error.

Line 100 checks location 222 to make sure that a `FILE LOCKED` error (error number 10) occurred. If it was not a `FILE LOCKED` error, the program goes to lines 200 and 210, which display the number of the error and the number of the line where it occurred.

A complete list of the ProDOS and Applesoft errors is in Appendix C.

If the file was locked, line 130 unlocks it, line 140 renames it, and line 150 locks it up again—under its new name.

Turning Off ONERR GOTO

On occasion you will use ONERR GOTO to detect one error, and it will be triggered by another error. To prevent this, you must turn off the ONERR GOTO feature. To do this, use the statement

```
POKE 216,Ø
```

Refer to the *Applesoft BASIC Programmer's Reference Manual* for more details on ONERR GOTO.

in your program when you want ONERR GOTO to be disabled. It is a good idea to use ONERR GOTO in the statement preceding the one that might generate the error, and to turn it off in the following statement.

Problems With ONERR GOTO

As described in the *Applesoft BASIC Programmer's Reference Manual*, ONERR GOTO does not work quite as it should. If you encounter apparent problems with ONERR GOTO, include the following lines in your program:

```
1  FOR I = Ø TO 9 : READ ZZ : POKE 768+I,ZZ
   : NEXT I

9999  DATA
1Ø4,168,1Ø4,166,223,154,72,152,72,96
```

Then, within your error handling routine, use the call

```
2ØØ  CALL 768
```

This ONERR fixing routine is more fully explained in the *Applesoft BASIC Programmer's Reference Manual.*

to activate this little ONERR fixer. Of course, you can change the numbers of any of these lines. Just make sure that line 1 is run before line 200 is.

I/O From BASIC Programs

The remainder of the commands in this chapter enhance your Apple II's ability to communicate with other BASIC programs and with devices such as printers and disk drives that are in the Apple II's peripheral connector slots. Such communication is referred to as the input of information and the output of information, collectively known as input/output, or more simply, I/O.

You can use the CHAIN command to let one program run another without destroying the variables that are currently in memory. Using CHAIN, two programs in separate files can use the same set of variables, or one very large program can be divided into more than one part.

With the STORE command you can save the names and values of all your program's variables. If a program stores all its variables before ending, the next time you run it you can use RESTORE to start up exactly where it left off.

You can use the PR# and IN# commands to let BASIC communicate with devices that are in any of the Apple II's slots.

Finally, you can use the FRE command to access ProDOS' fast housekeeping routines.

The CHAIN Command

If a BASIC program is too big to fit entirely in memory, you can use the CHAIN command to bring parts of the program into memory and then run each part, one at a time. All variables used, and all files opened (see Chapter 6) by the current part of the program are available to the parts of the program connected by the CHAIN command.

To run part of a program without throwing away the current variables or closing the open files, use the command

CHAIN pn [,@#] [,S#] [,D#]

When you chain from one part of a program to another, the first part of the program is removed from memory. To use the first part of the program again, use the CHAIN command again.

Execution of the second program begins at the line indicated by [,@#], the line number parameter. If you don't use [,@#], execution begins at the lowest numbered line in the program.

▲Warning A chained portion of the program cannot dimension an array used by a previous part of the program.

The Options

pn	pn indicates the file containing the BASIC program you want to run next.
[,@#]	If you use this option, the new program begins running at the line number specified by #. If the specified line does not exist, the next highest line in the program is run. If you omit this option, execution begins at the lowest numbered line in the program.
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

For Example

Here is an example of a program, PART1, that uses the CHAIN command with the line number option to connect a second program part, PART2. Both parts are already typed in for you. Set the prefix to /EXAMPLES/PROGRAMS using the command

```
PREFIX /EXAMPLES/PROGRAMS/
```

Look at PART1 and list it using the commands

```
LOAD PART1                                  Bring PART1 into memory.
LIST                                        Display it.
```

You see

```
5  REM PART1
10 D$=CHR$(4)                               D$ is (CONTROL)-(D).
20  PRINT D$;"PREFIX                        Set the prefix.
    /EXAMPLES/PROGRAMS"
30  I$="THE STRING I$ IS                    Set a string value.
    PRESERVED."
40  PRINT"PART1: I$ HAS BEEN SET."          And say that it's been set.
50  PRINT D$;"CHAIN PART2 ,@35"             Chain to line 35 of PART2.
```

The experiment here is to see if the variable I$ retains the value set by PART1 when it is printed out by PART2. Now type

---

```
LOAD PART2                                  Bring PART2 into memory.
LIST                                        Display it.
```

and you see

```
5  REM PART2
15  PRINT"PART2: WRONG LINE                 This shouldn't be printed.
    NUMBER."
25  GOTO 45                                 If it is, skip 35.
35  PRINT"PART2: RIGHT LINE                 This should be printed.
    NUMBER."
45  PRINT"PART2: ";I$                       And so should this.
```

If the CHAIN command in line 50 of PART1 works properly, line 35 should be the first line that is executed, and the statement `PART2: RIGHT LINE NUMBER.` should be printed on the screen. If the line number option does not work properly, the first line in the program (line 15) will be executed first, and the statement, `PART2: WRONG LINE NUMBER.` will be displayed.

Line 45 displays the line `PART2: THE STRING I$ IS PRESERVED` if the variables are preserved, and simply displays `PART2:` if the variable I$ is not preserved.

Place your bets on what will be printed out, and then type

```
RUN PART1
```

The STORE Command

The STORE command allows you to save to a disk file the names and values of all the variables that are used by a BASIC program. You can retrieve the variables using the RESTORE command.

An example using the STORE command is at the end of the section on the RESTORE command.

A game program, for example, can contain the STORE command to save the condition of a game when you stop playing. The next time you play the game, the program can RESTORE the variables from the file, and you can continue where you left off. You can also use the STORE command to create a set of information that is used by more than one program.

To store the current variables in a file use the command

STORE pn [,S#] [,D#]

The STORE command creates and places the variables in a file of type VAR.

▲Warning

Because ProDOS puts the variables in a compact form before it stores them, there may be a considerable time delay from when you issue the STORE command to when the disk drive starts spinning.

The Options

pn	pn is the pathname or partial pathname of the file in which to store the variables. If the file doesn't already exist, a file of type VAR is created.
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

The RESTORE Command

The RESTORE command allows you to get from a disk file the names and values of a set of variables to be used by a BASIC program. Only a file created by the STORE command can be retrieved by the RESTORE command.

To retrieve a set of variables from a file, use the command

RESTORE pn [,S#] [,D#]

This command clears all currently defined variables from memory before bringing in the new ones.

The Options

pn	pn is the pathname or partial pathname of the file containing the BASIC variables. The file must be of type VAR.
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

For Example

Your /EXAMPLES disk has a number guessing program called E.S.P. on it. Set the prefix to /EXAMPLES/PROGRAMS by typing

```
PREFIX /EXAMPLES/PROGRAMS
```

Then run the program by typing

```
RUN E.S.P.
```

Play with it for a while, and then press (Q) to quit. Type

```
RUN
```

again, and guess a number. Notice that the overall score starts at 0 again. You are going to use the STORE and RESTORE commands to make this program remember your overall score from one game to the next. Press (Q) to exit the game.

Display the program on the screen using the BASIC command

```
LIST
```

Your task is to add the STORE and RESTORE commands to the program. Since RESTORE clears all the variables that are currently defined, it is a good idea to use this command as the first line in a program. However, before including the RESTORE command in the program, you must create a file from which it can read variables.

Look at the last line in the program

```
200  END
```

This, the last line executed before the program ends, is the best place to place a STORE command. Type

```
200  PRINT D$;"STORE
     /EXAMPLES/DATA/ESPVARIABLES"
210  END
```

and then list the program again to be sure that the new lines are correct. Now when you run the program, the STORE command will create the new file ESPVARIABLES. Then you can add the RESTORE command to the program. Type

```
RUN
```

and play the game for as long as you like; then press (Q). Notice that the disk drive whirs as the variable file is placed on the disk. Type

```
CAT /EXAMPLES/DATA
```

to verify that the new file was created. Now look at the first few lines of the program by typing

```
LIST -30
```

Add the line

```
35  PRINT D$;"RESTORE
    /EXAMPLES/DATA/ESPVARIABLES"
```

Once again type

```
RUN
```

to play the game. Press any number, then look at the overall score. The game now remembers the total score of all the previous times you played the game. To save this game in the PROGRAMS directory, use the command

```
SAVE NEWESP
```

Do you have ESP?

The PR# Command

Your Apple II usually sends characters to the display screen. You can use the PR# command to change the destination of characters, sending them to a device in one of the Apple II's peripheral connector slots instead of to the screen. The syntax is

PR# snum

in which snum is a slot number from 0 to 7. For example, if your Apple II has an interface card for a printer installed in slot 1 (The Apple IIc can be addressed in this way as though there were a card in slot 1.), the command

```
PR# 1
```

causes subsequent printed characters to be sent to the printer. To restore the screen as the destination for printed characters, use the command

```
PR# 0
```

▲Warning If you are using an 80-column card (one is built into the Apple IIc), make sure it is turned off before you issue another PR# command. On an Apple IIe or IIc, press (ESC) (CONTROL)-(Q) to turn off the 80-column card. For other types of 80-column cards, refer to the card's documentation to discover how to turn it off.

Within a program, it is not necessary to turn off an 80-column card before a PR# command.

▲Warning Always remember to precede the PR# command with a (CONTROL)-(D) when you use it in a program. If you don't, ProDOS ignores the command entirely. If you think that your program isn't carrying out the PR# and IN# commands correctly, this could be the reason.

Starting Using PR#

If your Apple II has a disk controller card in one of its slots, you can start up the disk in that card's drive 1 with the command

PR# snum

in which snum is the number of the slot containing the card. (In the Apple IIc, the built-in disk drive is considered to be in slot 6, drive 1.) It may seem to you that this command only sets up the disk to receive future characters, but PR# actually does a little more.

When you use the PR# command to send output to a peripheral card in a slot, ProDOS tries to run the program in that card's read-only memory chip (most cards have them). The program in the ROM of a disk controller card automatically tries to read information from the disk; this is, of course, exactly what starting up the system is.

Refer to Chapter 9 for more details on using the PR# command to output characters.

You can also use the PR# command to call a machine-language program that is to perform the output of characters.

The Options

snum — snum is the number of the slot to which you want to write. If snum is in the range 1 to 7, inclusive, future characters are sent to the device in that slot. If snum is 0, future characters are sent to the screen. All other values of snum are invalid and must not be used.

For Example

First save any BASIC program that might be in memory, then place your /EXAMPLES disk in drive 1, close the door, and type

```
PR# snum
```

replacing snum with the number of the slot (probably 6) to which your Disk II controller card is connected. Disk drive 1 whirs and clicks and then ProDOS starts up as if you just turned the Apple II on.

If a printer is connected to one of the Apple II's slots, you can try this example too. First, turn on the printer. Then, replacing snum with the number of the slot to which your printer is connected, type

```
PR# snum
```

The printer make a little clicking noise. Like the disk controller card, the printer's card has a ROM chip that contains an initialization program. The printer card's program initializes the printer to a previously set condition, placing the head or printwheel to the beginning of the line, and doing whatever else needs to be done. Now type

```
CATALOG
```

and the contents of the /EXAMPLES volume directory are printed. Try out a few BASIC commands. You will find that everything that is normally printed to the screen is now printed on the printer. To return output of characters to the screen, type

```
PR# Ø
```

If your system has an 80-column card, you can now turn it back on.

The IN# Command

Your Apple II usually reads characters from the keyboard. You can use the IN# command to change the source of characters from the keyboard to a device in one of the Apple II's peripheral connector slots. The syntax of the command is

IN# snum

in which snum is a slot number from 0 to 7.

For example, if your Apple II has an external terminal connected through slot 3, the command

```
IN# 3
```

causes subsequent characters to be read from the terminal. To restore the Apple II's keyboard as the source for input characters, use the command

```
IN# Ø
```

▲Warning

Always remember to precede the IN# command with a CONTROL-D when you use it in a program. If you don't, ProDOS ignores the command entirely.

Refer to the section on PR# for more details on starting up. Chapter 9 contains an explanation of using IN# to input characters.

You can start up the disk in drive 1 of slot snum by typing the command

```
IN# snum
```

IN# can also be used to call a machine-language program that is to perform the character input operation.

The Options

snum	snum is the number of the slot from which you want to read. If snum is in the range 1 to 7, inclusive, future characters are read from the device in that slot. If snum is 0, future characters are read from the Apple II's keyboard. All other values of snum are invalid and must not be used.

The FRE Command

Use this command to access ProDOS' housekeeping routines.

For Example

Use FRE in immediate mode

```
FRE
```

or within a program

```
PRINT CHR$ (4);"FRE"
```

The BYE Command

This command allows you to leave Applesoft BASIC, and start another application program without going through the startup procedure referred to earlier in this chapter.

If you are using a hard disk and a program selector, BYE sends you the selector's menu. Otherwise, just type the pathname of the application program you want to use next.

For Example

Use BYE in immediate mode

```
BYE
```

or within a program

```
PRINT CHR$ (4);"BYE"
```

Chapter 6 Text in Files

About This Chapter

This chapter introduces you to the use of ProDOS text files. It describes how to create them, how to place information in them, and how to take information from them.

The first part of the chapter is an introduction to the two types of text files: sequential-access text files, and random-access text files. The next part of the chapter teaches you how to write programs that use sequential-access text files. The last part of the chapter is a description of the text file commands as used with sequential-access text files.

The next chapter teaches you how to write programs that use random-access text files. It is a continuation of this chapter, so read this chapter first.

You might use the commands described in this chapter

- in a program that keeps a list of words for a guessing game
- in a program that saves and retrieves text
- in a program that saves and analyzes experimental data.

This Chapter's Commands

The commands in this chapter are summarized below. These are all the commands you need to use ProDOS text files.

OPEN: Prepare to use a file

You must use this command before you use a text file. If the file mentioned does not exist, a text file is created. If the file does exist, OPEN checks to see that the file is a text file.

CLOSE: Stop using a file

Use this command to tell ProDOS that you have finished reading from and writing to a file. Before ending, your program must close all the files that it opened.

WRITE: Prepare a file for writing

Use this command to tell ProDOS the file you want to write to and where in the file you want to start writing. You can use the WRITE command only after the file is opened; it remains in effect until you give the next ProDOS command.

READ: Prepare a file for reading

Use this command to tell ProDOS the file you want to read and where in the file you want to start reading. You can use the READ command only after the file is opened; it remains in effect until you give the next ProDOS command.

APPEND: Prepare to write to the end of a file

Use this command to write data starting at the end of a text file. It opens the file, positions to the end of the file, and then writes to the file.

FLUSH: Send all unwritten data to the file

ProDOS writes characters to files in groups, not one by one. FLUSH causes all characters that are not yet written to a file to be sent. After you use FLUSH, you can be sure that the characters in the file are identical to those that the program has printed. The CLOSE command does a FLUSH before it actually closes a file.

POSITION: Read and discard fields in a file

A field is a sequence of characters that ends with a carriage return.

This command lets you skip a specified number of fields in the text file before you read or write more information.

Note

Of these commands, only CLOSE and FLUSH can be used in immediate mode. All can be used in programs.

Sequential-Access Text Files: An Introduction

In this section, a *scroll* models a sequential text file.

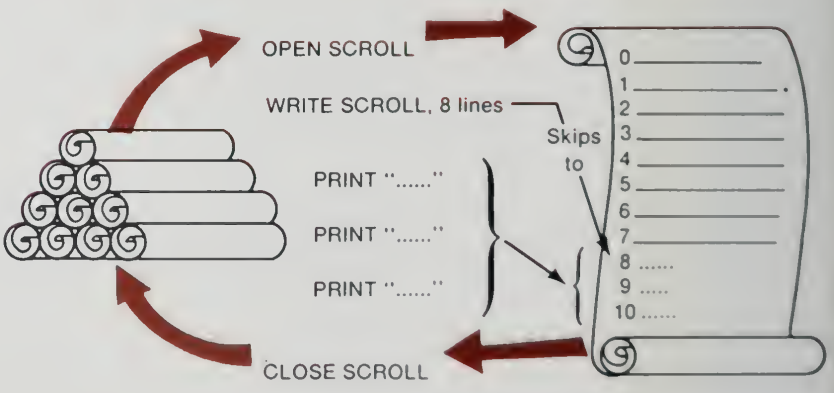
You can think of a disk full of sequential-access text files as a collection of scrolls. Each scroll, like the sequential text file it models, can contain an unlimited number of lines of text. The analogy is appropriate because in both cases you must search through line by line to locate a particular line of text—there are no pages or markers to make the search faster.

As you read these rules, bear in mind that a scroll represents a sequential text file, the scroll's name represents the file's name, and a line on the scroll represents one line, or field, of text within the text file. A field is simply a string of characters that ends with a carriage return. There is also a pointer to keep track of your current position in the scroll.

To print new lines onto a scroll, use these commands in this order:

1. OPEN name. This selects the named scroll, opens it, and points the pointer to the first line. If a scroll by that name is not in the collection, one is created. You must use OPEN before you can read from or write to a scroll.
2. WRITE name [,number of lines]. This starts at the pointer of the named scroll, and skips lines, one by one, until it has skipped number of lines. You must use WRITE before you can use PRINT (step 3).
3. PRINT phrase. This places phrase on the line pointed to. Phrase can be a character, a number, a word, or an entire line. Phrases are printed one after another unless you use WRITE to select a new line number. PRINT destroys anything that was previously on the line. You can repeat this step as often as necessary.
4. CLOSE name. This rolls the scroll back up, and returns it to the collection.

Figure 6-1. Printing to a Scroll



Here are the commands you use to read lines from a scroll:

1. OPEN name. This selects the named scroll, opens it, and points the pointer to the first line. If a scroll by that name is not in the collection, one is created. You must use OPEN before you can read from or write to a scroll.
2. READ name [,number of lines]. This starts at the pointer of the named scroll, and skips lines, one by one, until it has skipped number of lines. You must use READ before you can INPUT phrases from the scroll (step 3).
3. INPUT phrase. This reads a phrase from the current line of the scroll. If there are no more phrases on the current line, it reads the first phrase from the next line. You can repeat this step as often as necessary.
4. CLOSE name. This rolls the scroll back up, and returns it to the collection.

You can have up to eight scrolls simultaneously open. That is why you must always refer to them by name.

The phrase used with the PRINT and INPUT statements can be any expression or list of expressions allowed by BASIC.

Random-Access Text Files: An Introduction

In this section, a *notebook* models a random-access text file.

You can think of a disk containing random-access text files as a collection of notebooks. Each notebook, like the text file it models, has a name and an unlimited number of pages. Each of a notebook's pages holds the same number of characters, but since lines can be of differing lengths, there is no specific number of lines on a page.

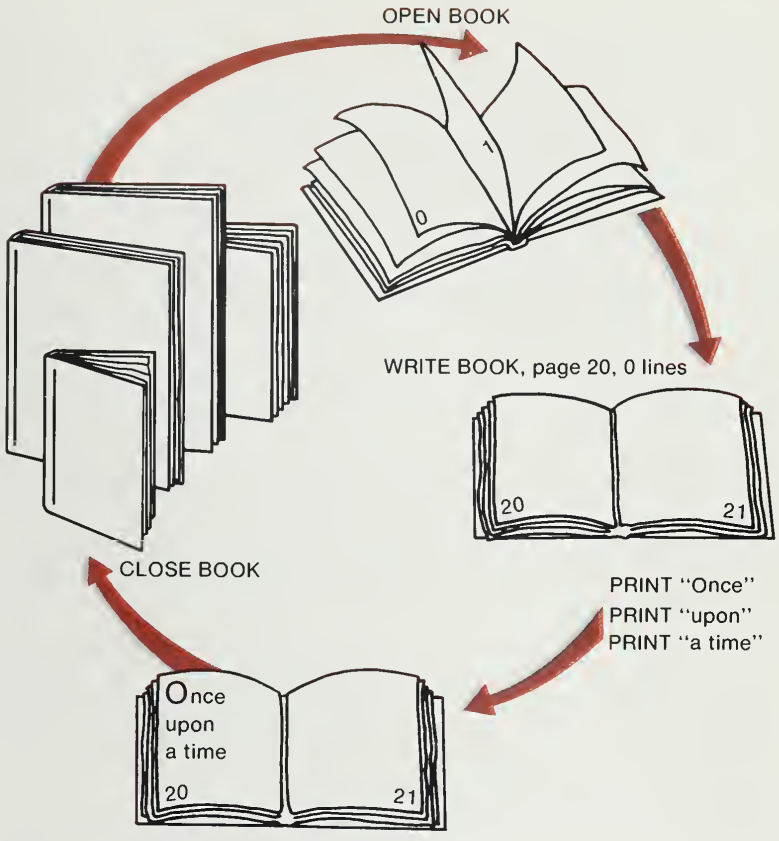
This analogy is appropriate because in both cases you can *flip* to a certain page before reading or writing lines of text.

As you look through these rules, remember that a notebook represents a random-access text file; the notebook's name represents the file's name; each page in the notebook represents one record in the file (each record in a file holds the same number of characters); and a line on a page represents a field in a record. There is also a pointer to keep track of your current position on the current page of the notebook.

To write information on a page of a notebook, you use these commands in this order:

1. OPEN name. This selects the named notebook, opens it, and points the pointer to the first line of the first page. If no notebook by that name exists, one is created. You must use OPEN before you can read from or write to a notebook.
2. WRITE name [,page number] [,number of lines]. This quickly opens the named notebook to page number. If the page with that number is not yet in the notebook, that page is added to the notebook. If a number of lines is given, that many lines are skipped, one by one. You must use WRITE before you can use PRINT (step 3).
3. PRINT phrase. This adds phrase to the current line on the current page. Phrase can be a character, a number, a word, or a line. Phrases are placed one after another until you use WRITE again, so you must be careful not to print past the end of the page. You can repeat this step as often as necessary.
4. CLOSE name. This closes the named notebook and returns it to the collection.

Figure 6-2. Printing to a Notebook



To read from a page in one of your notebooks, use these commands in the following order:

1. OPEN name. This selects the named notebook, opens it, and points the pointer to the first line of the first page. If no notebook by that name exists, one is created. You must use OPEN before you can write to or read from a notebook.
2. READ name [,page number] [,number of lines]. If you use page number, this quickly opens the named notebook to page number. If a number of lines is given, that many lines are skipped, one by one. You can next use INPUT to read from that page.
3. INPUT phrase. You read each phrase from the page with an INPUT statement. Phrases and characters are read sequentially from the page until you use READ again, so you must be careful not to read past the end of the page. You can repeat this step as often as necessary.
4. CLOSE name. This closes the named notebook and returns it to the collection.

You can have up to eight notebooks open at any given time. That is why you must always refer to them by name.

A random-access text file has an unlimited number of **records** (pages) each holds a fixed number of characters. Within each record, you can print as many fields (lines) as will fit. As mentioned above, a **field** is a string of characters that ends with a carriage return.

Sequential- and Random-Access Text Files

As the scroll analogy illustrates, you can use the information in a sequential-access text file in a sequential manner only, that is, starting at the beginning of the file and working towards the end. Because of this, sequential files are best suited for applications that read the entire contents of the file at the beginning of the program, and that write the modified contents back to the file at the end of the program.

The records (pages of a notebook) in a random-access text file, however, can be used in any order; a program can modify a single record of the file without affecting the others. Thus random-access text files are best for programs that keep track of a large number of pieces of information that are about the same size.

So how do you decide which type of text file to use? It is a matter of preference, but you might want to consider the following aspects of text file use:

Disk space: The first time you write to a record in a random-access text file, the entire record is placed on the disk. Thus if your records are each 200 characters in size, and if you write only one character to each of them, you are wasting 199 characters of disk space per record. Because records aren't usually entirely filled, random-access text files use up more disk space than do sequential text files.

Amount of data: If you are going to read all the information into memory at the beginning of the program, it is faster to read it, field by field, from a sequential text file.

Use of data: If the information won't all fit in memory, and you won't use it in any particular order, it is much faster to use a random-access text file.

Sequential text files are best for lists of variable length information, such as lists of words or lines of text. In fact, many word processors store their text in sequential text files. Later in the chapter you will write programs that place text in, and read text from, sequential text files.

Random-access text files are best for storing many pieces of information that are of the same size, and that will change frequently. You might use random-access files to store monthly inventory records, a list of names and addresses, or even a file of help screens (the text for each ProDOS help screen is stored in one record of a random-access text file). You will write a program that uses random-access files to keep a list of names and addresses.

Position-in-the-File Pointer

In the scroll and notebook analogies there was a pointer that kept track of the current position. Every open text file has one too. As you read from a file, the current position is the character following the last character read. Likewise, when you write to a file, the current position becomes the spot in the file immediately following the last character written. When you first open a file, the pointer indicates the first character position in the file.

The current position is the character following the last read or written character.

In the rest of this manual, the position-in-the-file pointer is referred to as the **current position**.

To see a complete description of the text file commands and their options, refer to Appendix B, the summary of ProDOS commands.

The text file commands have many options. Because you will use a few of them most of the time, and most of them only once in a while, the commands and their options are explained by example, with emphasis on the most frequently used options.

Work through the examples in the order given. Explanations of concepts that have already been presented will be brief.

The Field

The basic unit of a sequential text file is a field. A field, like a line of text on the screen, is a series of characters that ends with a carriage return character. When you print a line to the screen using the BASIC statement PRINT, without a terminating semicolon, the line is ended with a carriage return, and the cursor goes to the next line. Likewise, when you print to a file using the BASIC statement PRINT, without a terminating semicolon, the field is terminated with a carriage return; subsequent printed characters go into the next field in the file.

The following BASIC statement could be used to write a line to the screen, or a field to a text file.

```
50 PRINT "THIS CREATES A FIELD"
```

The following line, however, writes just part of a line to the screen, or part of a field to a file; the semicolon at the end prevents a carriage return character from ending the current field.

```
100 PRINT "THIS MAKES PART OF A FIELD";
```

A subsequent PRINT statement adds characters to the same line on the screen, or to the same field in a text file. A sequential text file can contain any number of fields.

Storing Characters in Fields

Here is an example that shows the way characters are stored in the fields of a sequential text file. Assume that you have already given the OPEN and WRITE commands. You can place six fields in an open sequential text file using these BASIC statements.

```
40  PRINT "GREEN"
50  PRINT "YELLOW"
60  PRINT "ORANGE"
70  PRINT "RED"
80  PRINT "VIOLET"
90  PRINT "BLUE"
```

A program would normally use the CLOSE command to close the file. Here is how the characters generated by lines 40 through 90 would be stored in a file. A carriage return is represented by the symbol >.

```
GREEN>YELLOW>ORANGE>RED>VIOLET>BLUE>
{ F0 }{  F1 }{  F2 }{F3}{  F4 }{F5 }
```

This sequential text file has six fields, and contains 36 characters.

Note The first field in a sequential text file is field number 0.

A Simple Sequential Text File

To create a new sequential text file, use the OPEN command with a filename that does not yet exist. Here is a short program that places each of its lines in a sequential text file.

```
5  REM PROGRAM LISTSELF
10 D$ = CHR$ (4)                    D$ is (CONTROL)-(D).
20  PRINT D$;"PREFIX                Set the prefix to indicate the
    /EXAMPLES/DATA"                 /EXAMPLES/DATA directory.
30  PRINT D$;"OPEN LISTFILE"        Open LISTFILE; create it if it doesn't yet
                                    exist.
40  PRINT D$;"WRITE LISTFILE"       Prepare LISTFILE for writing.
50  LIST                            Put the program's listing in LISTFILE; line 40
                                    directs it there.
60  PRINT D$;"CLOSE"                CLOSE all open files.
```

This program is very simple. It opens LISTFILE (line 30), uses WRITE so that LISTFILE can be written to (line 40), and then gives the BASIC command LIST. Notice that LIST is not a ProDOS command, and is not preceded by a (CONTROL)-(D). Because the WRITE command redirects output to a file, the LIST command places the lines of the program, one by one, into the sequential text file named LISTFILE, rather than on the screen. The last line of the program closes the file.

Type in the program, and with the /EXAMPLES disk in a drive, type

```
RUN
```

The disk drive whirs as the text file is placed on the disk. When a new prompt appears on the screen, type

```
CAT /EXAMPLES/DATA
```

and look for LISTFILE in the DATA directory.

EXEC command: see Chapter 8.

How can you check to see what is in the new file? Here's a little secret. The EXEC command tells your Apple II to take commands from a sequential text file rather than from the keyboard. When you type in lines of a BASIC program from the keyboard, they are entered as a BASIC program. Thus, if you use the EXEC command to enter lines of a program from a sequential text file, they too must be entered as a BASIC program. Type

```
NEW
```

to remove the program from memory. Now type the command

```
EXEC /EXAMPLES/DATA/LISTFILE
```

One prompt symbol appears on the screen for each line in the BASIC program. When the disk stops spinning, and the prompts stop prompting, type

```
LIST
```

and you'll see that the program has reappeared in memory. The EXEC program is described in Chapter 8; there you will see that this technique of listing a program to a file is a valuable tool.

Writing to a File Using PRINT

A carriage return is represented by the symbol > .

There are several ways that the PRINT command can be used to place characters into a text file. In the examples in Table 6-1 `A$` has the value `DOG`, and `B$` has the value `CAT`.

Table 6-1. Printing to a Text File

PRINT Statement	Adds Characters	Comments
`PRINT "TEXT"`	`TEXT>`	Completes current field.
`PRINT "TEXT";`	`TEXT`	Adds to current field.
`PRINT A$;B$;`	`DOGCAT`	Adds to current field.
`PRINT A$,B$`	`DOGCAT>`	Unlike PRINT to the screen, spaces are not added between elements separated by commas. Completes current field.
`PRINT A$;",";B$`	`DOG,CAT>`	Adds two elements to the current field, and completes the field. See second example, Table 6-2.

The examples in Table 6-1 are intended to show three basic techniques: adding characters to the current field, completing the current field (subsequent characters will go to the next field), and adding elements to a field.

Elements, each written and read by a single variable, are strings of characters that are separated by commas. They deserve special mention because you may need to use some special techniques to retrieve them intact from files.

Reading Characters From a File

There are several ways to read characters from a text file: INPUT is best for some types of data, and GET is better for others. Table 6-2 shows some of the ways to read characters.

Table 6-2. Reading From a Text File

Input Statement	Effect
`INPUT A$`	Reads one element of a field. If there is more than one element in the field, the rest of the field is discarded.
`INPUT A$,B$`	Reads two elements of a field. If there are more than two elements in the field, the rest of the field is discarded. If there are not two elements in the field, elements are read from the next field.
`GET C$`	Reads the next character from the file. The GET statement reads all characters, including commas and colons. This is a good way to read fields with varying numbers of elements.

As illustrated by the first two examples in Table 6-2, an INPUT statement must contain one variable for each element in a field if it is to read all the elements from the field. If a carriage return is read before characters are assigned to all variables, characters are automatically taken from the next field.

One Element Per Field

Here is a program that writes four fields, each containing one element, to a sequential text file.

```
5  REM MAKE.FRUIT
10 D$ = CHR$ (4)                          D$ is (CONTROL)-(D).
20  PRINT D$;"PREFIX                      Set the prefix to /EXAMPLES/DATA.
    /EXAMPLES/DATA"
30  PRINT D$;"OPEN FOUR.FRUITS"           Create the file FOUR.FRUITS. if necessary,
                                          and OPEN it.
40  PRINT "THIS GOES TO THE               Before WRITE is used, characters
    SCREEN"                               still go to the screen.
50  PRINT D$;"WRITE FOUR.FRUITS"          Prepare FOUR.FRUITS for writing.
60  PRINT "APPLE"                        Put field 0 in the file.
70  PRINT "BANANA"                       Put field 1 in the file.
80  PRINT "CANTALOUPE"                   Put field 2 in the file.
90  PRINT "DATE"                         Put field 3 in the file.
100  PRINT D$;"CLOSE FOUR.FRUITS"         Close FOUR.FRUITS.
```

Notice that even after FOUR.FRUITS is open, you can still PRINT to the screen (line 40). However, after the WRITE statement in line 50, all PRINT statements send their characters to the file. Here is how the characters are stored in FOUR.FRUITS.

The symbol > means (RETURN).

```
APPLE>BANANA>CANTALOUPE>DATE>
{ F0 }{ F1  }{   F2    }{ F3}
```

Here is a program that reads the four fields out of the sequential text file FOUR.FRUITS, and into successive elements of an array. It also prints them onto the screen so you can see that it's working.

```
5  REM  GET.FRUIT
10 D$ = CHR$ (4)                      D$ is (CONTROL)-(D).
20 R$ = CHR$ (13)                     R$ is Carriage Return.
30  PRINT D$;"PREFIX                 Set the prefix to /EXAMPLES/DATA.
    /EXAMPLES/DATA"
40  PRINT D$;"OPEN FOUR.FRUITS"       Prepare FOUR.FRUITS for use.
50  PRINT D$;"READ FOUR.FRUITS"       Prepare FOUR.FRUITS for reading.
60  FOR I = 0 TO 3                   For fields 0 through 3,
70 : INPUT A$(I)                       read field I from the file,
80 : PRINT A$(I)                       and print it on the screen.
90  NEXT I                            Then do the next field.
100  PRINT D$;"CLOSE FOUR.FRUITS"     Close FOUR.FRUITS when done.
```

This program uses the INPUT statement once for each field it reads from the file. If you wanted to read all four fields with a single INPUT statement, you could replace lines 60 through 90 with

```
60  INPUT A0$,A1$,A2$,A3$             When there are no elements left in a field,
                                      INPUT reads from the next field.
70
80
90  PRINT A0$,R$,A1$,R$,A2$,R$,A3$    Print the four fruits on the screen, one fruit
                                      per line.
```

The string variable R$ was set to carriage return in line 20. When it is printed, a carriage return is printed on the screen.

Multiple Elements Per Field

Here is a program that places three elements in each of two fields. Following it are programs that read the elements in different ways.

```
5  REM CONJUGATE
10 D$ = CHR$ (4)                    D$ is (CONTROL)-(D).
20  PRINT D$;"PREFIX                Set the prefix to
    /EXAMPLES/DATA"                 /EXAMPLES/DATA.
30  PRINT D$;"OPEN VERB.LIST"       Create the file VERB.LIST, if necessary, and
                                    prepare to use it.
40  PRINT D$;"WRITE VERB.LIST"      Prepare to write to VERB.LIST.
50  PRINT "DRINK,DRANK,DRUNK"       Print three elements in field 0.
60  PRINT "THINK,THANK,THUNK"       Print three elements in field 1.
70  PRINT D$;"CLOSE VERB.LIST"      Close VERB.LIST.
```

The commas between the verbs in lines 50 and 60 are actually written to the file. When an INPUT statement with multiple variables reads these commas, it treats them as markers for the end of the element currently being read. Here is the character sequence for the file VERB.LIST (the symbol > means (RETURN)).

```
DRINK,DRANK,DRUNK>THINK,THANK,THUNK>
{        F0        }{        F1        }
```

Here is a program that reads each verb from the file into a separate variable.

```
5  REM CONJUGEATEN
10 D$ = CHR$ (4)                    D$ is (CONTROL)-(D).
20 R$ = CHR$ (13)                   R$ is Carriage Return.
30 C$ = ","                         C$ is Comma.
40  PRINT D$;"PREFIX                Set the prefix to
    /EXAMPLES/DATA"                 /EXAMPLES/DATA.
50  PRINT D$;"OPEN VERB.LIST"       Prepare to use VERB.LIST.
60  PRINT D$;"READ VERB.LIST"       Prepare to read from VERB.LIST.
70  INPUT A1$,A2$,A3$,A4$,A5$,A6$   Read three elements from field 0, and three
                                    elements from field 1.
80  PRINT A1$;C$;A2$;C$;A3$;R$;     Print all six elements
          A4$;C$;A5$;C$;A6$           on two separate lines.
90  PRINT D$;"CLOSE VERB.LIST"      Close VERB.LIST.
```

Notice that line 70 simply reads consecutive elements from the file. When all the elements are read from one field, elements are automatically taken from the next.

Line 80 prints out the elements so that they look just like they did in the original program. Note the use of C$ to print a comma, and R$ to print a carriage return.

The next program reads only the present tense verbs.

```
5  REM CONJUGEAT
10 D$ = CHR$ (4)                          D$ is (CONTROL)-(D).
20 R$ = CHR$ (13)                         R$ is Carriage Return.
30  PRINT D$;"PREFIX                      Set the prefix to
    /EXAMPLES/DATA"                       /EXAMPLES/DATA.
40  PRINT D$;"OPEN VERB.LIST"             Prepare to use VERB.LIST.
50  PRINT D$;"READ VERB.LIST"             Prepare to read from VERB.LIST.
60  INPUT A1$                             Read first element from field 0.
70  INPUT A2$                             Read first element from field 1.
80  PRINT A1$;R$;A2$                      Display the two verbs.
90  PRINT D$;"CLOSE VERB.LIST"            Close VERB.LIST.
```

In this example, as explained in Table 6-2, each INPUT statement causes an entire field to be read, regardless of the number of elements used in the INPUT statement. Thus line 60 causes the string "DRINK to be assigned to A1$, and line 70 causes the string "THINK" to be assigned to A2$. Finally, the verbs DRINK and THINK are displayed.

GET Characters From a File

Data element: A string of characters separated by commas.

The INPUT statement has its limitations. It is designed to read data elements—strings of characters separated by commas. If you want to read in strings of characters that may contain commas, colons, or other control characters, or if you want to detect particular characters as they are read, you must use the GET statement to read the characters one by one.

The GET statement works the same whether you are reading information from the keyboard or from a text file.

You can use the GET statement to read a variable number of elements from a field. If you have been working through the examples, you have already used the program CONJUGATE to create the file VERB.LIST. If you haven't done this, use the command

```
RUN /EXAMPLES/PROGRAMS/CONJUGATE
```

and a text file named /EXAMPLES/DATA/VERB.LIST is created.

Although we know that there are three elements in each field of VERB.LIST, there are situations in which your program will not know how many elements to read from a field. Let's write a subroutine that uses the GET statement to read any number of elements, separated by commas, from one field of a file.

The following subroutine reads elements, separated by commas, and then places them into consecutive elements of string array A$. The element in use at any time is indicated by A$(I).

The GET statement returns one character. This subroutine reads a character into the variable C$, and if the character is not a comma or a carriage return, adds it to A$(I). Because a comma separates two elements, the subroutine upon reading a comma adds 1 to the variable I, causing I to indicate the next element of the array, and then continues reading characters. It repeats this process until it reads a carriage return, which marks the end of the field.

```
20 R$ = CHR$ (13)                          R$ is Carriage Return.
999  REM READ A FIELD
1000 I = 0                                 Start with array element 0.
1010 I = I + 1                             Use next array element.
1020  GET C$                               Read the next character.
1030  IF C$ = "," THEN GOTO 1010           If comma, use next element.
1040  IF C$ = R$ THEN RETURN               If carriage return, you're done.
1050  A$(I) = A$(I) + C$                   Otherwise, add C$ to element.
1060  GOTO 1020
```

The next program uses this subroutine to retrieve the elements from the first field of the file VERB.LIST. Recall that you use a subroutine by saying GOSUB followed by the number of the line on which the subroutine starts (see line 60 below). When the RETURN statement in the subroutine is executed (line 1040 above), the line following the GOSUB statement is executed (line 70 below).

```
5  REM USE SUBROUTINE
10 D$ = CHR$ (4)                    D$ is (CONTROL)-(D).
30  PRINT D$;"PREFIX                Set the prefix to
    /EXAMPLES/DATA"                 /EXAMPLES/DATA.
40  PRINT D$;"OPEN VERB.LIST"       Prepare VERB.LIST for use.
50  PRINT D$;"READ VERB.LIST"       Prepare VERB.LIST for reading.
60  GOSUB 1000                      Read all elements from a field.
70  PRINT D$;"CLOSE VERB.LIST"      Close VERB.LIST.
80  FOR J = 1 TO I                  Now print the I elements of A$ onto the
                                    screen.
90  : PRINT A$(J)
100  NEXT J
110  END
```

To test this program, type in the lines of the program and the lines of the subroutine, and then type

```
RUN
```

You see the three words in the first field of the file VERB.LIST printed on the screen, one word per line.

In this example only the first field was read from the file. A technique for reading a variable number of fields from a file is in the program GET.TEXT below.

Entering and Reading Text

You are now going to write two short programs: one that reads text from the keyboard and then saves it to a file, and another that reads text from a file and then prints it on the screen.

As you read through the following explanations of the programs, don't bother typing in the lines of the programs. You can find these programs in the /EXAMPLES/PROGRAMS/ directory as the files MAKE.TEXT and GET.TEXT.

A Program for Entering Text

This program is stored in the file /EXAMPLES/PROGRAMS/MAKE.TEXT. It lets you type in up to one hundred lines of text and save them in a file. It asks for lines of text, reads them from the keyboard, and places them into consecutive elements of the array A$. The program stops reading lines as soon as it encounters an empty line. This portion of the program is:

```
10  DIM A$(100)
110 I = I + 1: PRINT I;": ";

120  GOSUB 1000

130  IF A$(I) <  > "" GOTO 110
```

Allow enough room for 100 lines of text.
Ask for next line of text with a line number followed by a colon.
Call a subroutine that reads a line of characters into array element A$(1).
If A$(1) is not empty, then go to line 110 which prompts for the next line of text.

This part of the program uses a couple of little tricks. Since the value of a variable is 0 the first time it is used, the first time line 110 is executed, I is set to 1, and the prompt

```
1:
```

is printed on the screen. Next, line 120 reads a line of text into array element I. Thus the 1th line of text is placed in array element I. If that array element is not empty—that is, if it does not have a length of zero—then line 130 goes to line 110, which asks for the next line of text.

Once all the text is in the array, the program asks you for the name of the file in which it is to place the text:

```
140  PRINT D$;"PREFIX /EXAMPLES/DATA"
150  INPUT "SAVE TO WHAT FILE? ";N$
160  IF LEN (N$) = 0 THEN GOTO 230
```

and reads that name into the variable N$. Line 160 gives you a chance to end the program without saving the text to a file. If the name of the file has a length of zero—that is, if you pressed RETURN before typing any other characters—the program goes to line 230 which is the end of the program. (Notice the different techniques used by lines 160 and 130 to detect empty strings.)

Next comes the task of saving the contents of the array to the named file. The last line that the program read in is line I, and it is an empty one (used to indicate the end of the text). Therefore, now print lines 1 through I - 1 to the file. Do this as follows:

```
20 D$ = CHR$ (4)                    D$ is (CONTROL)-(D).
170  PRINT D$;"OPEN ";N$            Open the named file (N$).
180  PRINT D$;"WRITE ";N$           Prepare to write to the file.
190  FOR J = 1 TO I - 1             For each line of text,
200 : PRINT A$(J)                     print the line to the file
210  NEXT J                           and continue to the next line.
220  PRINT D$;"CLOSE ";N$           Close the file when done.
```

Finally, end the program with the line

```
230  TEXT : END
```

The purpose of the TEXT command is explained below.

If you type in the program as presented so far, it works. The following lines print the instructions for using the program onto the screen, and freeze them there.

```
40  TEXT : HOME
50  PRINT "              TEXT FILE CREATOR"
60  INVERSE : PRINT "TO ADD A LINE:"
70  NORMAL : PRINT "    ENTER CHARACTERS,
    AND PRESS RETURN"
80  INVERSE : PRINT "TO END:"
90  NORMAL : PRINT "    PRESS RETURN ON AN
    EMPTY LINE"
100  PRINT : POKE 34,6
```

Line 40 causes the Apple II to switch the display to full screen text mode, to clear all characters from the screen, and to move the cursor to the upper-left corner of the screen. Lines 50 through 90 place the instructions for the program on the screen, some in normal letters, and some in inverse letters. Line 100 freezes the upper six lines of the screen so they remain on the screen even if you enter more lines than the screen can hold. This is called **setting the text window**.

Setting the text window is discussed in an appendix of the *Applesoft BASIC Programmer's Reference Manual.*

Run this program a few times, creating text files of different lengths. Experiment with the different features of the program to become familiar with the way they work.

For Example: You can enter a blank line of text by putting spaces on that line; because the line contains characters, the length of the line is not 0.

A Program for Retrieving Text

This program is in the file /EXAMPLES/PROGRAMS/GET.TEXT. Load the program so you can look at the lines as they are described below.

First the program sets up the variables it is going to use. It dimensions the array A$ to hold up to 100 elements, and it assigns the values RETURN and CONTROL-D to R$ and D$, respectively.

```
20  DIM A$(100)
30 D$ = CHR$ (4)
40 R$ = CHR$ (13)
```

Allow enough room for 100 lines of text.
D$ is CONTROL-D.
R$ is Carriage Return.

In line 60, the program asks for the name of the file from which it is to read text, and reads the filename into the variable N$.

```
50  TEXT : HOME
55  PRINT D$;"PREFIX /EXAMPLES/DATA"
60  INPUT "LIST WHAT TEXT FILE? ";N$
```

Having read the name of the text file, the program can now read consecutive lines from the file:

```
100  PRINT D$;"OPEN ";N$
110  PRINT D$;"READ ";N$
120  FOR I = 1 TO 100
130 : PRINT I;": ";: GOSUB 1000

140  NEXT I
150  PRINT : PRINT D$;"CLOSE ";N$
160  TEXT : END
```

Open the text file.
Prepare to read from the file.
For each line that could be in the text file,
  read the line into array element I
  and then print it on the screen.
Then do the next line.
Close the file.
Restore text mode, and end.

There is a small problem with this part of the program. If the text file contains fewer than 100 lines, the program reads the last line of the text file, and then has no more characters to read. The program tries to read a character anyway, fails, and prints out the error message

```
END OF DATA

BREAK IN 130
```

To prevent this occurrence, use the ONERR GOTO statement described in Chapter 5. If you include the line

```
10 ONERR GOTO 150
```

the program, upon encountering an error, simply closes the file (line 150) and ends.

Note

This is not the best way to use the ONERR GOTO statement. Before taking any action, a better version of this program would check to see which error occurred. Refer to the section on the ONERR GOTO statement in Chapter 5 to see how to do this.

The subroutine that reads lines of text is, as before

```
1000  GET C$: PRINT C$
1010  IF C$ = R$ THEN RETURN
1020  A$(I) = A$(I) + C$
1030  GOTO 1000
```

The lines presented so far comprise a complete program; if you type them in, they will run. As before, you can add a few lines that permanently place instructions for using the program on the screen.

```
70  PRINT : PRINT "TYPE:  CONTROL-S   TO
    STOP LISTING"
80  PRINT "       ANY KEY      TO CONTINUE"
90  PRINT : PRINT : POKE 34,6
```

Line 50 causes the Apple II to switch to full screen text mode, to clear all characters from the screen, and to move the cursor to the upper-left corner of the screen. Lines 70 and 80 place instructions for the program on the screen. Line 90 freezes the upper six lines of the display so they remain on the screen even if you enter more lines than the screen can hold.

You can use the OPEN command only in deferred mode.

Before your program can write to or read from a sequential text file, it must open the text file using this command

OPEN pn [,S#] [,D#]

Files that you open must be closed (usually at the end of the program). If you don't close a file that you open and write to, you may lose some of the written data.

When a program opens a text file, ProDOS designates a space in memory, called the **file buffer**, to hold all important information about the file, and prepares the system to read or write starting at the beginning of that file. Up to eight files can be open at once.

If the file designated by pn does not yet exist, a file with that name is created, and is added to the proper directory. If the file exists, and is already open, you get the **FILE BUSY** error. You (or the program) must close the file before opening it again.

▲Warning When you open a file, the pathname or partial pathname with which you opened the file becomes that file's *identifier*. In all subsequent references to that file, you must use exactly the same pn, even if you change the prefix. See the example that follows.

Assume the prefix is currently set to /APPLE/ (a popular fruit). If you open the file /APPLE/STRUDEL using the BASIC line

```
10  PRINT D$;"OPEN STRUDEL"
```

you must always use the name STRUDEL when referring to that file. For example, if you subsequently use the commands

```
20  PRINT D$;"PREFIX /STRAWBERRY/"
30  PRINT D$;"WRITE /APPLE/STRUDEL"
```

you get the **FILE NOT OPEN** error. Even though the prefix has changed, you should still use the same file identifier:

```
30  PRINT D$;"WRITE STRUDEL"
```

The OPEN command has other options that are not applicable to sequential text files. They are discussed elsewhere.

The Options

pn	pn indicates the name of the file to be opened. If the file already exists, it must not be open. If the file does not yet exist, a file of type text (TXT) is created.
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

For Example

Several examples of the OPEN command have already been presented. The following section explains a new aspect of OPEN.

Delete Before Opening

Suppose your program routinely replaces an old text file with a new one with the same name. If the new one is shorter than the old one, then unless the program deletes the old file first, the new one has part of the old file hanging on the end. If you don't want all this extra text at the end of the file, you must delete the old file before writing to the new one. This is usually easy, but what if it is the first time you have run the program, and the *old file* doesn't yet exist?

For example, suppose a game creates and uses the file /GAMES/DINGER , and you wish your program to delete that file at the start of each new game. The line

```
950  PRINT D$;"DELETE /GAMES/DINGER"
```

causes the error message

```
FILE NOT FOUND
```

if the file doesn't exist, and the program halts.

Here's a quick way to delete a file and open it for new data, whether or not the file already exists:

```
10 D$ = CHR$ (4)                    D$ is (CONTROL)-(D).
20 N$ = "/GAMES/DINGER"             N$ is the name of the file.
30  PRINT D$;"OPEN ";N$             Open the file. If it does not already exist, it is
                                    created.
40  PRINT D$;"CLOSE ";N$            Close the file before deleting it.
50  PRINT D$;"DELETE ";N$           Since the file definitely exists, the file can be
                                    deleted.
60  PRINT D$;"OPEN
```

The rest of the program goes here.

Simply open and close the file before deleting it. This ensures that the file exists and can be deleted.

The CLOSE Command

You can use the CLOSE command in either immediate or deferred mode.

After a program finishes writing to or reading from a file, it must close the file. Proper closure of every file is necessary to ensure that all characters are written to their files, and that the file buffers are properly released. CLOSE takes the form

CLOSE [pn]

▲Warning A program must always close a file that it opened. In some circumstances, however, a program that contains an error will stop before it can close all open files. When this happens, issue the CLOSE command from the keyboard to close all open files.

The Options

The CLOSE command without any options closes all open files.

[pn] pn indicates the name with which the file was opened.

The WRITE Command

You can use the WRITE command in deferred mode only.

You must use the WRITE command before you can use the PRINT statement to place characters in a file. The WRITE command identifies to ProDOS the file to which you want to write characters, and the position in the file where the first character will be placed. The WRITE command remains in effect until the next ProDOS command is given. This command takes the form

WRITE pn [,F#] [,B#]

The Options

Use the F# and B# options to choose the position of the first character to be written to the file. If you don't use these options, the first character is written to the file's current position.

pn	pn indicates the name of the file to be written to. It must be identical to the name with which the file was opened.
[,F#]	# is the number of fields past the current position that ProDOS is to read and discard. ProDOS does this by reading characters, starting at the current position, until it has read the specified number of carriage returns. This option changes the file's current position.
[,B#]	# is the number of bytes, or characters, that ProDOS must read and discard. The new current position is the sum of # and the previous current position.

Note

You can temporarily turn off READ and WRITE, without changing the position in the file, by using

```
PRINT CHR$ (4)
```

Re-issuing the READ and WRITE then continues operation on the file at the last position.

This would allow you, for example, to write part of a file, query the user for a response, and then resume writing.

The READ Command

You can use the READ command only in deferred mode.

You must use the READ statement before you can use the INPUT and GET statements to read characters from a file. The READ command identifies to ProDOS the file from which to read characters and the position in the file from which to read the first character. The READ command remains in effect until the next ProDOS command is given. This command takes the form

READ pn [,F#] [,B#]

The Options

Each time you use the READ command you must identify a file by name (pn).

Use the F# and B# options to choose the position of the first character to read from the file. If you don't use these options, the first character is read from the file's current position.

pn	pn indicates the pathname or partial pathname of the file you want to read from. It must be identical to the value of pn with which the file was opened.
[,F#]	# is the number of fields which ProDOS is to read and discard. ProDOS does this by reading characters, starting at the current position, until it reads the specified number of carriage returns. This option changes the file's current position.
[,B#]	# is the number of bytes, or characters, that ProDOS is to read and discard. The new current position is the sum of # and the previous current position.

The APPEND Command

Use the APPEND command only in deferred mode.

See the descriptions of the OPEN, POSITION, and WRITE commands for more information on the operation and use of the APPEND command.

The APPEND command lets you add data to the end of a sequential text file. It is like three commands in one: it opens the file (see the OPEN command), positions to the end of the file (see the POSITION command), and then writes to that file (see the WRITE command). This command has the form

APPEND pn [,S#] [,D#]

After giving the APPEND command, your program can send data to the file using the PRINT command.

The Options

pn	pn indicates the file to be appended. It must not be open. If the indicated file does not yet exist, the file is created.
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

For Example

You can modify the program MAKE.TEXT—which creates a sequential text file—so it adds lines to the end of a text file.

With the /EXAMPLES disk in drive 1, and with ProDOS started up, load /EXAMPLES/PROGRAMS/MAKE.TEXT into memory and display it on the screen with the commands

```
LOAD /EXAMPLES/PROGRAMS/MAKE.TEXT
LIST
```

Do you remember how MAKE.TEXT works? First it reads lines of text into the array A$, then it asks you for a filename, and then it opens the file and prints the lines of text to the file.

To append lines of text to the end of a file, all you need to do is replace the OPEN and WRITE statements with an APPEND statement. Replace lines 50, 170, and 180 by entering the following lines

```
50  PRINT "            TEXT FILE APPENDER"
170  PRINT D$;"APPEND ";N$
180
```

If you want to keep this program, you can save it by typing

```
SAVE /EXAMPLES/PROGRAMS/APPEND.TEXT
```

The APPEND command, like the OPEN command, creates a new file if the file you try to append does not already exist.

The FLUSH Command

Use the FLUSH command in either immediate or deferred mode.

As a program writes to a text file, ProDOS stores a block of 512 bytes, or characters, of data before any of the data is placed on the disk. If you use the FLUSH command, all the characters that are currently stored are transferred to the file. After you use the FLUSH command, you can be sure that every character written to a file is actually placed in that file. The FLUSH command takes the form

FLUSH [pn]

The Options

The FLUSH command without any options flushes all open files.

[pn] pn indicates the file to be flushed. It must be identical to the pn with which the file was opened.

For Example

This command can be useful if you want to make a program absolutely foolproof. If you use the FLUSH command after each statement that prints to a file, you can be sure that every character actually reaches the file. Your programs will be a little longer and a little slower, but a lot more reliable.

This command is also of great value for data collection applications in which there are frequent power outages. If your application program is named STARTUP, the program will restart after each power outage. Using FLUSH, you can maximize the amount of data collected.

Note | Frequent use of the FLUSH command slows down your program. You must decide the importance of speed versus data integrity.

The POSITION Command

Use the POSITION command in deferred mode only.

With the POSITION command, you can access the information in any field or byte within a file. This command, which takes the form

POSITION pn,F#

starts at the current position, and reads and discards the number of fields mentioned in F#. The file must be open.

For example, if the current position is within the fourth field of a file, and you want to read from the tenth field in the file, skip six fields using the POSITION command with the option ,F6.

The Options

pn	pn indicates the file whose current position is to be altered. It must be identical to the pn with which the file was opened.
,F#	# indicates the number of fields to be read and discarded. If you try to position past the end of the file, you get the `END OF DATA` error message.

Chapter 7 Random-Access Text Files

About This Chapter

This chapter introduces you to the use of ProDOS random-access text files. It describes how to create them, how to place information in them, and how to take information out of them.

Because random-access text files are so similar to sequential-access text files, this chapter assumes that you are already familiar with the material in the preceding chapter.

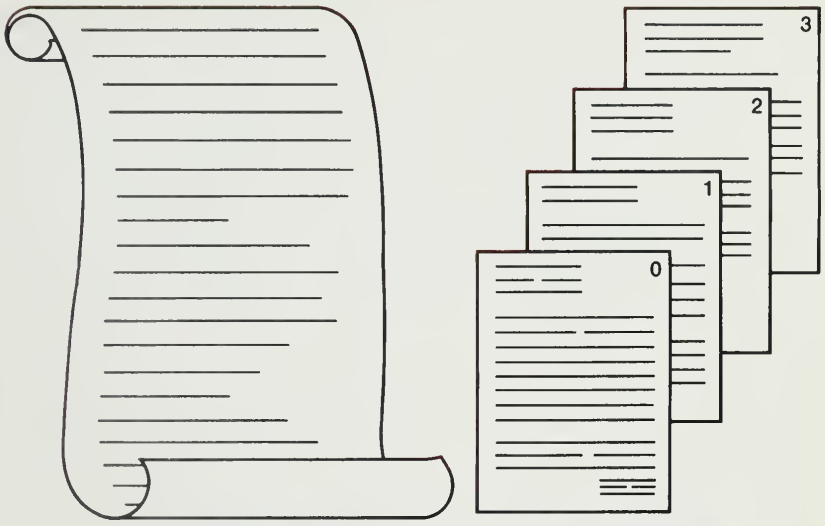
The first part of this chapter explains the structure of random-access text files and how you can make use of it. The next part of the chapter leads you through a typical program that uses random-access text files. The remainder of the chapter contains a description of the commands you use to manipulate random-access text files.

By the Way: Notice that although the commands described in this chapter are the same as those in Chapter 6, the options that accompany them are different. Appendix A contains a summary of the commands with all their options.

Random-Access Text Files

As illustrated by the scroll versus notebook analogy in Chapter 6, there is a fundamental difference between sequential and random-access text files: a sequential text file is a single unit, composed of a series of fields; a random-access text file consists of multiple units, or records, all the same size, each composed of a series of fields. Figure 7-1 illustrates this comparison.

Figure 7-1. Sequential and Random-Access Text Files



Record Length

Record length is the number of characters a record can hold. All records in a single random-access text file are the same length.

When you open a random-access text file for the first time, you must assign it a record length. For example, to open a random-access text file named /MUSIC/CLASSICAL that has a record length of 33, use the command

```
30  PRINT D$;"OPEN /MUSIC/CLASSICAL ,L33"
```

The length of a record is the number of characters it can hold, including the carriage return. Each record in the file /MUSIC/CLASSICAL is 33 characters long.

Notice that when you open a random-access text file, you don't need to specify the number of records that the file is going to hold. ProDOS takes care of such details for you.

Writing to a Record

When you use the WRITE command with a random-access text file, you must specify the number of the record to which you are going to write (just like writing to a page of a notebook). If the specified record does not yet exist, ProDOS reserves enough space on the disk for that entire record. Thus, even if you are only going to place one character in a record of the CLASSICAL file, that record will use 33 characters' worth of disk space.

For example, before writing to record 10 of the file you just opened, use the command

```
50  PRINT D$;"WRITE /MUSIC/CLASSICAL ,R10"
```

The subsequent PRINT statements you use place characters into record 10 of this file.

Inside a Record

Storing characters is explained in the section on fields in Chapter 6.

The storage of characters in a record is just like the storage of characters in a sequential text file. The difference is that there is a maximum number of characters that will fit into each record.

▲Warning

You must be careful not to print more characters to a record than it can hold. If you do, ProDOS simply prints the extra characters into the beginning of the next record. For this reason, a file's record size must be at least as great as the largest number of characters to be stored in any of its records. Don't forget that the carriage return at the end of each field is counted as a character too.

Reading From a Record

When you use the READ command with a random-access text file, you must specify the number of the record from which you want to read. For example, to read the seventh record from the CLASSICAL file, use the command

```
90  PRINT D$;"READ /MUSIC/CLASSICAL,R7"
```

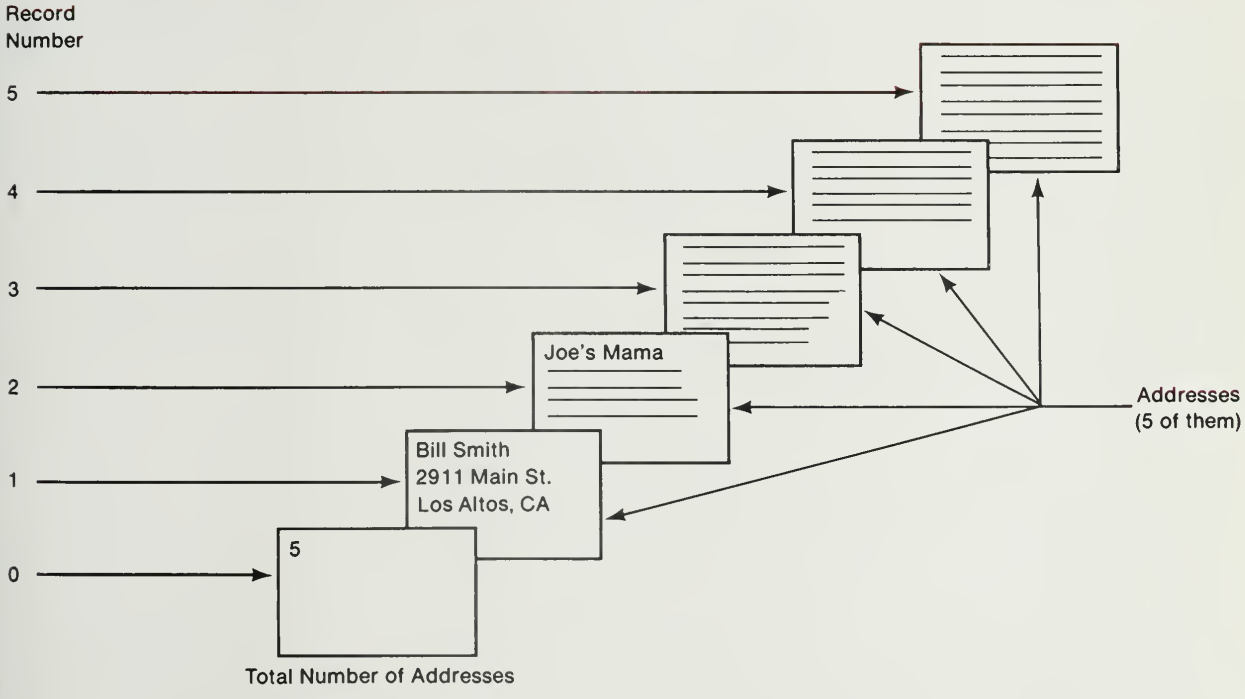
followed by the appropriate INPUT or GET statements. If the record does not exist, the READ statement is allowed, but the first INPUT or GET statement causes the `END OF DATA` error message.

A Sample Program

To illustrate the use of random-access text files, here is a short program that you can use to keep track of an address list. The program has two main tasks: to enter new addresses and to look up addresses that are already entered. Each of these two parts is a subroutine; a main program calls these subroutines as needed.

The first address you give the program is stored in record one of the file, the second in record two, and so on. The total number of records in the file is stored in record zero of the file. Figure 7-2 represents the structure of the file when it contains five addresses.

Figure 7-2. Five Addresses in the File



The total number of addresses in the file is initially 0. First write and run a little program that places the initial total number in record 0.

```
5  REM CREATE FILE, WRITE RECORD Ø
1Ø  D$ = CHR$ (4)                          D$ is (CONTROL)-(D).
2Ø  PRINT D$;"OPEN                         Open the file.

/EXAMPLES/DATA/BLACK.BOOK,L2ØØ"
3Ø  PRINT D$;"WRITE                        Write to record 0.
    /EXAMPLES/DATA/BLACK.BOOK,RØ"
4Ø  PRINT Ø                                Put a 0 there.
5Ø  PRINT D$;"CLOSE"                       Close all files.
```

Line 20 opens the file BLACK.BOOK, in which addresses are to be kept, with a record length of 200. Thus, addresses stored by the program can have no more that 200 characters in them. Line 30 specifies that data will be written to record 0 of the file. Line 40 prints a 0 to that record, and line 50 closes the file.

Type in this program and run it. The disk drive spins as the new file BLACK.BOOK is created (OPEN causes a new file to be created if it does not already exist), and the 0 is placed in it. Then type

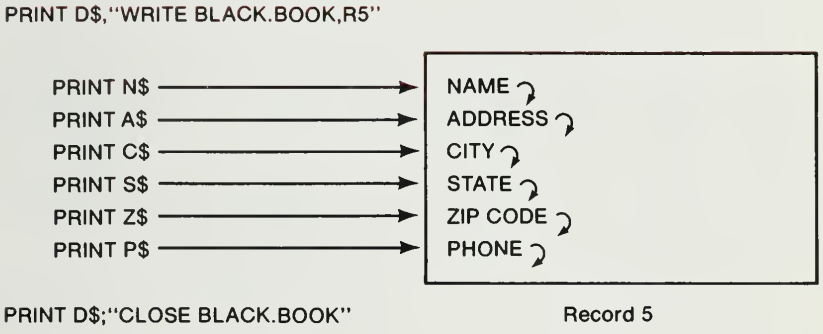
```
NEW
```

to remove this program from memory. You no longer need it.

Writing a Record

You must now decide how information is arranged within each record. Each record contains a name, an address, a city, a state, a zip code, and a phone number. Place each piece of information in a separate field; that is, use a separate print statement to place each piece in the record. To improve the clarity of the program, use a separate variable for each piece of the address. Figure 7-3 shows the BASIC statements that place an address in record five of a file named BLACK.BOOK.

Figure 7-3. Writing an Address to Record Five



To write a new record to a file:

1. Read in the new address to be entered. Store it in the variables N$,A$,C$,S$,Z$,P$.
2. Add 1 to the total number of records stored (TR=TR+1). This is the number of the new record.
3. Use OPEN and WRITE to prepare that record to be written.
4. Print the new information to the file.
5. Print the new total number to record 0.
6. Close the file.

Here is a subroutine that does this. Lines 1010 through 1060 gather the information for an address.

```
1000  REM  READ NEW INFO
1010  INPUT "NAME:          ";N$
1020  INPUT "ADDRESS:       ";A$
1030  INPUT "CITY:          ";C$
1040  INPUT "STATE:         ";S$
1050  INPUT "ZIP CODE:      ";Z$
1060  INPUT "PHONE:         ";P$
```

Read name to be entered.
Read address.
Read city.
Read state.
Read zip code.
Read phone number.

Lines 1070 through 1120 open the file whose name is stored in F$ and write information to a new record (record TR) in that file.

```
1070  PRINT D$;"OPEN ";F$;",L200"
1080  PRINT D$;"READ ";F$;",R0"
1090  INPUT TR:TR = TR + 1
1100  PRINT D$;"WRITE ";F$;",R";TR
1110  PRINT N$: PRINT A$: PRINT C$
1120  PRINT S$: PRINT Z$: PRINT P$
```

Open the file with record length of 200.
Read total records.
Get total records, add 1.
Prepare to write to record number TR.
Place each part of address in a separate field.

Lines 1130 through 1160 print to record 0 the total number of address records that are now in the file, close the file, and end the subroutine.

```
1130  PRINT D$;"WRITE ";F$;",R0"
1140  PRINT TR
1150  PRINT D$;"CLOSE ";F$
1160  RETURN
```

Prepare to write record number in record 0.
Print new record number.
Close the file.
End of subroutine.

These lines are already typed in and stored as part of the program in the file /EXAMPLES/PROGRAMS/ADDRESS. If you want to test just this subroutine, load /EXAMPLES/PROGRAMS/ADDRESS, and add the lines:

```
11 F$ =
   "/EXAMPLES/DATA/BLACK.BOOK"
12  GOSUB 1000
13  GOTO 12
```

Store records in file /EXAMPLES/DATA/BLACK.BOOK.
Call the subroutine, over and over again.

When you want to stop the program, press CONTROL-C and then press RETURN. These keystrokes, as described in the *Applesoft Tutorial*, stop almost any program. If you typed in lines 11-13, type

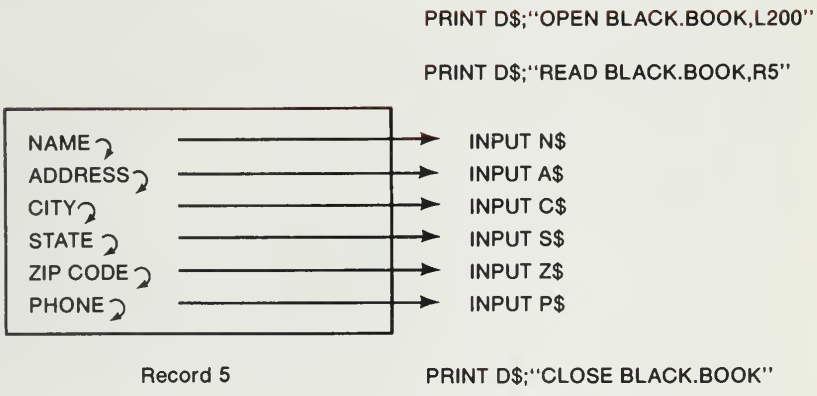
```
11
12
13
```

to remove them from the program.

Reading a Record

You already know how information is stored in the records. Your new task is to find a way to ask which record (address) is to be displayed. To do this, display the name from each record, together with its record number, and then ask for the number of the desired address.
Figure 7-4 shows the BASIC statements that read an address from record five of the file BLACK.BOOK.

Figure 7-4. Reading an Address From Record Five



To read a record from the file:

1. Use OPEN and READ to prepare the file to be read.
2. Read the total number of entries from the file.
3. Print a numbered list of the entries on the screen.
4. Find out which entry to display.
5. Print the selected record on the screen.
6. Close the file.

Here is a subroutine that does all these things. First it reads the total number of addresses from record 0 of the file. Then it reads the name from each address and prints all of them on the screen.

```
2000  PRINT D$;"OPEN ";F$;",L200"
2010  PRINT D$;"READ ";F$;",R0"
2020  INPUT TR
2030  IF TR = 0 THEN GOTO 2210
2040  HOME
2050  PRINT "WHOSE ADDRESS DO YOU
      WANT? ": PRINT
2060  FOR I = 1 TO TR
2070 : PRINT D$;"READ ";F$;",R";I
2080 : INPUT N$
2090 : PRINT I,N$

2100  NEXT I
```

Open the file.
Read from record 0.
Get the number of records.
Check for no records.
Clear the screen.
For each record I, position to record I, and read the stored name.

Print the record number and the name on the screen.

Repeat for all records.

Because the READ statement causes ProDOS to read characters from the file rather than from the keyboard, you must cancel the READ statement's action before reading anything from the keyboard. A READ or a WRITE is canceled by a ProDOS command; one way of doing this is by using the empty ProDOS command in line 2110 below.

Line 2120 asks for a number and then reads it into the string R$. Because the question asks for a numeric answer, it could have read the answer into a regular variable, but then the accidental pressing of a letter key would cause an error. Line 2130 converts R$ to a number (R), and the next line compares the number to the valid address numbers. An invalid response causes the question to be redisplayed.

```
2110  PRINT D$: PRINT                          Empty ProDOS command; turns off READ.
2120  INPUT "TYPE A NUMBER AND
      PRESS RETURN";R$
2130 R = VAL (R$)                              Get numeric value of answer.
2140  IF R < 1 OR R > TR THEN GOTO             If bad number, try again.
      2120
```

The last part of this subroutine places an address on the screen. It uses READ to position to the requested record and uses INPUT to read the six fields from the record. Then it prints the six fields on the screen. Line 2230 prevents the address from being erased before you have a chance to read it.

```
2150  HOME                                     Clear the screen.
2160  PRINT D$;"READ ";F$;",R";R               Prepare to read record R.
2170  INPUT N$,A$,C$,S$,Z$,P$                  Read address.
2180  PRINT N$: PRINT A$                       Print name and address.
2190  PRINT C$, PRINT S$                       Put city and state on same line.
2200  PRINT Z$: PRINT P$                       Print zip code and phone.
2210  PRINT D$;"CLOSE ";F$                     Close the file.
2220  VTAB 23: HTAB 8                          Position the cursor.
2230  INPUT "PRESS RETURN TO                   Preserve the screen.
      CONTINUE";T$
2240  RETURN
```

Controlling the Program

The remainder of the program is simple. Here is the main part of the program that lets you choose between entering a new address, reading an existing address, or ending.

```
10 D$ = CHR$ (4)                          D$ is (CONTROL)-(D).
20 F$ =                                   F$ is the file of addresses.
   "/EXAMPLES/DATA/BLACK.BOOK"
30  HOME                                  Clear the screen.
40  PRINT "WHAT WOULD YOU LIKE TO
    DO?": PRINT
50  PRINT "  1  ENTER A NEW
    ADDRESS"
60  PRINT "  2  LOOK UP AN
    ADDRESS"
70  PRINT "  3  END": PRINT
80  INPUT "TYPE A NUMBER AND PRESS
    RETURN ";C$
90 C = VAL (C$)                           Convert response to number.
100  IF C < 1 OR C > 3 THEN GOTO          If bad entry, try again.
     30
110  IF C = 1 THEN GOSUB 1000             Enter a new address.
120  IF C = 2 THEN GOSUB 2000             Look up an address.
130  IF C = 3 THEN END
140  GOTO 30
```

Lines 80 through 100 use the same technique that you used earlier to choose a record to be displayed. They read in a letter, convert it to a number, and then check to see that the number falls in the expected range. If it is a bad entry, the program repeats the question. A good entry causes the proper subroutine to be called.

See if you can modify the program to delete and change entries.

The OPEN Command

Use the OPEN command in deferred mode only.

Before a program can write to or read from a random-access text file, it must open the text file using this command

OPEN pn [,L#] [,S#] [,D#]

When you open a random-access text file for the first time, you must open it with the length option, L#. # is the number of bytes, or characters, that each record can hold. Each subsequent time you open the file, the original record length is assumed.

If you open a random-access text file with a record length other than that with which it was created, the new record length is used as long as the file is open, but the original record length remains as the default.

Files that are opened must be closed, usually at the end of the program. If you don't close a file that you open and write to, you may lose some of the written data.

When a program opens a text file, ProDOS designates a space in memory, called the **file buffer**, to hold all important information about the file. It also prepares the system to read or write starting at the beginning of the file. Up to eight files can be open at a time.

▲Warning ProDOS uses the name with which you opened the file as the file's identifier. Always use the exact same name, even if you change the prefix.

pn	pn indicates the name of the file to be opened. If the file already exists, it must not be open. If the file does not yet exist, it is created as a text file.
[,L#]	You must use the length option the first time you open the file, that is, when the file is created. If you create a file without the length option, the file is given a record length of one. The record length, #, must be in the range 1 to 65535.

▲Warning Be sure that a file's record length is greater than the maximum number of characters that you will ever put in one of the file's records. If you accidentally write past the end of a record, data is written onto the beginning of the next record in the file, destroying any data that might be in that next record.

[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

The CLOSE Command

Use the CLOSE command in either immediate or deferred mode.

After a program finishes writing to or reading from a file, it must close the file. Proper closure of every file is necessary to ensure that all characters are written to their files, and that the file buffers are properly released. CLOSE takes the form

CLOSE [pn]

▲Warning

A program must always close a file that it opened. In some circumstances, however, a program contains an error and stops before it can close all open files. When this happens, issue the CLOSE command from the keyboard to close all open files.

The Options

The CLOSE command without any options closes all open files.

[pn] pn indicates the name of the file to be closed. It must be identical to the name with which you opened the file.

The WRITE Command

Use the WRITE command only in deferred mode.

You must use the WRITE statement before you can use the PRINT statement to place characters in a record of a random-access file. The WRITE command tells ProDOS the file, the number of the record, and the position within the record of the first character to be written. The WRITE command remains in effect until another ProDOS command is given. This command takes the form

WRITE pn [,R#] [,F#] [,B#]

You must use the WRITE command each time you want to write to a record other than the current record. If you use the WRITE command without the R# option, ProDOS defaults to record 0.

The Options

pn	pn indicates the file to be written to. It must be identical to the name with which the file was opened.
[,R#]	# is the number of the record to which characters are to be sent. If this option is omitted, record 0 is assumed. The maximum record number is 16 megabytes divided by the file's record length, or 65535, whichever is smaller. If # is larger than any previous record number, the **ENDFILE** column in the catalog changes. Refer to section "The End of File" for more details.
[,F#]	# is the number of fields that ProDOS is to read and discard. ProDOS does this by reading characters, starting at the current position, until it has read the specified number of carriage returns. This option changes the file's current position.
[,B#]	# is the number of bytes, or characters, that ProDOS is to read and discard. This new current position is the sum of # and the previous current position.

The End of File

Each file listed by the CATALOG command has a number under the **ENDFILE** heading. For all types of files except random-access, this number indicates the number of bytes in the file. For random-access files, this number represents the number of bytes that would be in the file if every record, from 0 to the highest numbered record written to, were used.

For example, assume that a random-access text file has a record length of 50, and that data is written to records 0, 156, 756, and 1890. This file has data stored in four blocks on the disk. However, **ENDFILE** is calculated by multiplying the total possible number of records by the record length (1891 * 50 = 94550). This number is used by ProDOS to determine the location on the disk of the last record in the file—needed, for example, if you want to append a record to the file.

The READ Command

Use the READ command only in deferred mode.

Before you can read from a record, you must use the READ command to indicate the number of the record you want to read.

This command takes the form

READ pn [,R#] [,F#] [,B#]

When used with random-access text files, the READ command tells ProDOS the file from which the next INPUT or GET statements will take characters (pn), the record within the file from which characters are to be read (R#), and the position within the record from which the first character is to be read (F# and B#). The READ command remains in effect until the next ProDOS command is given.

The Options

pn	pn indicates the name of the file to be read. It must be identical to the name with which you opened the file.
[,R#]	# is the number of the record from which you are going to read. If you don't use this option, record 0 is assumed. If R# indicates a record that doesn't exist, you don't get an error; the first INPUT or GET statement from a non-existent record causes an error. The maximum allowable record number is 16 megabytes divided by the record length or 65535, whichever is smaller.
[,F#]	# is the number of fields past the beginning of the indicated record which ProDOS is to read and discard. ProDOS does this by reading characters, starting at the current position, until it has read the specified number of carriage returns. This option changes the file's current position.
[,B#]	# is the number of bytes, or characters, that ProDOS is to read and discard. The new current position is the sum of # and the previous current position.

The APPEND Command

Use the APPEND command in deferred mode only.

You can use the APPEND command to add data to the end of a random-access text file. It is like three commands in one: it opens the file, positions to the beginning of the record that follows the last record in the file, then it writes to that file. This command has the form

APPEND pn [,L#] [,S#] [,D#]

After giving the APPEND command, your program can send data to the file using the PRINT command.

The Options

pn	pn indicates the file to be appended. It must be open. If the indicated file does not yet exist, it is created.
[,L#]	# indicates the length of the file's records. If # is the same as the record length assigned when the file was created, the next character written is the first character following the last record in the file. If not, see the following description. First ProDOS positions to the last character in the file (the character indicated in the ENDFILE column in a CATALOG). It divides ENDFILE by the record length; the remainder is the offset into the current record. Finally it uses ENDFILE to find the position of the first character in the next record (ENDFILE—Offset + Record Length + 1). Thus, if you use a record length of 1 when appending to a random-access text file, the next character written is the character following ENDFILE; that is, it is the same as appending a sequential text file.
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

For Example

You can change the ADDRESS program so it uses APPEND in the subroutine that writes new records.

With the /EXAMPLES disk in drive 1, and with ProDOS started up, load ADDRESS into memory and display part of it on the screen with the commands

```
LOAD /EXAMPLES/PROGRAMS/ADDRESS
LIST 1070,1160
```

You now see these lines:

```
1070  PRINT D$;"OPEN ";F$;",L200"        Open the file with record length of 200.
1080  PRINT D$;"READ ";F$;",R0"          Read total records.
1090  INPUT TR:TR = TR + 1               Get total records, add 1.
1100  PRINT D$;"WRITE ";F$;",R";TR       Prepare to write to record number TR.
1110  PRINT N$: PRINT A$: PRINT C$       Place each part of address in a separate field.
1120  PRINT S$: PRINT Z$: PRINT P$
1130  PRINT D$;"WRITE ";F$;",R0"         Prepare to write record number in record 0.
1140  PRINT TR                           Print new record number.
1150  PRINT D$;"CLOSE ";F$               Close the file.
1160  RETURN                             End of subroutine.
```

The purpose of lines 1070 through 1100 is to discover the number of the last record, and to prepare to write to it. The APPEND command does just that. Replace those four lines with

```
1070  PRINT D$;"APPEND";F$;",L200"       Add after last record.
1080
1090                                     Delete these lines.
1100
```

After the subroutine writes the new address to the record (lines 1110 and 1120), it saves the new number of records in record 0 of the file. Before the program can do this it must discover the total number of records, as was previously done in lines 1080 and 1090. Just place the same two lines somewhere between 1120 and 1130 (say, 1124 and 1126), as shown below, and the revision of the subroutine is complete. It should now look like this:

```
1070  PRINT D$;"APPEND";F$;",L200"      Put new address at end.
1110  PRINT N$: PRINT A$: PRINT C$      Place each part of address in a separate field.
1120  PRINT S$: PRINT Z$: PRINT P$
1124   PRINT D$;"READ ";F$;",R0"        Read total records.
1126   INPUT TR:TR = TR + 1             Get total records, add 1.
1130  PRINT D$;"WRITE ";F$;",R0"        Prepare to write record number in record 0.
1140  PRINT TR                          Print new record number.
1150  PRINT D$;"CLOSE ";F$              Close the file.
1160  RETURN                            End of subroutine.
```

The FLUSH Command

Use the FLUSH command in either immediate or deferred mode.

As a program writes to a text file, ProDOS stores a block of 512 bytes, or characters, of data before any of the data is placed on the disk. If you use the FLUSH command, all the characters that are currently stored are transferred to the file. After you use the FLUSH command, you can be sure that every character written to a file is actually placed in that file. The FLUSH command takes the form

FLUSH [pn]

The Options

The FLUSH command without any options flushes all open files.

[pn] pn indicates the file to be flushed. It must be identical to the name with which the file was opened.

The POSITION Command

Use the POSITION command only in deferred mode.

When used with random-access text files, the POSITION command works exactly as it does with sequential text files. You can use it to skip fields within the current record. This command takes the form

POSITION pn [,F#] [,B#]

Starting at the current position, ProDOS reads and discards the numl of fields specified by F#, then it reads and discards the number of bytes specified by B#. The file must be open.

For example, if the current position is within the fourth field of a record, and you want to read from the tenth field in that record, ski six fields using the POSITION command with the option ,F6.

▲Warning | Because it doesn't check record size, this command can skip across record boundaries.

The Options

pn	pn indicates the file whose position is to be altered.
,F#	# indicates the number of fields to be read and discarded. If you try to position past the end of file, you get the `END OF DATA` error message
[,B#]	After skipping the number of fields specified by the field option, ProDOS reads and discards the number of bytes, or characters, specified by # in the byte option.

Chapter 8 EXEC: Control From a Text File

About This Chapter

This chapter explains how you can use the EXEC command to cause the Apple II to take its commands from a sequential text file rather than from the keyboard. This sequential text file can contain ProDOS commands, lines of BASIC program, or even lines of input to a BASIC program—nearly any command that you can type from the keyboard.

Because the various uses of EXEC are not always obvious, examples are given to show how the EXEC command can be used. This chapter provides several diverse examples; this may give you some ideas for using it in new and different ways.

Throughout this chapter, a text file that is to be used with the EXEC command is called an EXEC file, and the contents of an EXEC file are called an EXEC program.

Note Because EXEC files are externally identical to all other text files, you'll find it useful to place the ending *.EXEC* on all EXEC filenames. This convention is used throughout this chapter.

You can use an EXEC file

- to give automatically a frequently-used set of commands
- to use instead of repeatedly typing the same inputs into a program
- to combine BASIC programs or subroutines
- to put a machine-language routine in a BASIC program.

This chapter first gives you a demonstration of the EXEC command, then a few different ways to create EXEC files, and finally some sample applications of the EXEC command. The section that describes the EXEC command in detail is at the end of the chapter.

EXEC Demonstration

There are two steps to the EXEC demonstration. In the first step you run a BASIC program that creates an EXEC file, and in the second step you use the EXEC command to tell your Apple II to take its commands from the EXEC file.

With your /EXAMPLES disk in drive 1, set the prefix by typing

```
PREFIX /EXAMPLES/PROGRAMS/
```

and then type

```
RUN EXEC.DEMO
```

to see this page of instructions:

```
      << EXEC DEMONSTRATION >>

THIS PROGRAM CREATES A SEQUENTIAL TEXT
FILE NAMED "SHOWOFF.EXEC".  EACH STRING
THAT IS PLACED IN SHOWOFF.EXEC BY THIS
PROGRAM IS A LEGAL APPLE II COMMAND.

WHEN YOU TYPE
     EXEC SHOWOFF.EXEC
THE COMMANDS IN THE FILE SHOWOFF.EXEC
TAKE CONTROL OF YOUR COMPUTER.  EACH
COMMAND IS EXECUTED JUST AS IF IT HAD
BEEN TYPED IN FROM THE KEYBOARD.
"BASIC PROGRAMMING WITH PRODOS"
DESCRIBES THE PROGRAM IN MORE DETAIL.

         << HAPPY EXECUTING >>

PRESS THE SPACE BAR TO MAKE THIS
PROGRAM CREATE THE FILE SHOWOFF.EXEC.
PRESS Q TO STOP THIS PROGRAM NOW.
```

Press (SPACE). The screen goes blank, the message

```
CREATING SHOWOFF.EXEC ...
```

appears on the screen, and the disk drive's disk use light glows as the program writes the SHOWOFF file onto the disk. When the program finishes, you see this message

```
IT'S DONE!!!

YOUR APPLE IS READY TO SHOWOFF A LITTLE
BIT.  ALL YOU HAVE TO DO IS TYPE

     EXEC /EXAMPLES/PROGRAMS/SHOWOFF.EXEC

PRESS THE RETURN KEY, AND ENJOY.

OH, BY THE WAY.  IF YOU WANT TO SEE THE
COMMANDS IN THE FILE SHOWOFF, USE
GET.TEXT FROM CHAPTER 6.  TYPE

     RUN GET.TEXT

AND WHEN IT ASKS FOR A FILENAME, JUST
TYPE IN

     /EXAMPLES/PROGRAMS/SHOWOFF.EXEC
```

Before you actually run the EXEC program, take a look at SHOWOFF.EXEC using the GET.TEXT program from Chapter 6. Type

```
RUN GET.TEXT
```

then choose the file SHOWOFF.EXEC. Notice the wide variety of commands, that can all be typed from the keyboard. Set this command file into action by typing

```
EXEC /EXAMPLES/PROGRAMS/SHOWOFF.EXEC
```

As SHOWOFF.EXEC is running, it describes everything it is doing. Surprised? Don't be; showoffs are hardly ever modest.

Create an EXEC File Using BASIC

A BASIC program that creates an EXEC file must:

1. Use the OPEN command to open the text file.
2. Use the WRITE or APPEND command to prepare the file to be written to.
3. Use the PRINT or LIST command to place commands in the text file.
4. Then use the CLOSE command to close the text file.

Printing the Commands to the File

Here is a step-by-step example that illustrates how to create an EXEC file named DOIT.EXEC that contains these commands

```
PREFIX /EXAMPLES/PROGRAMS/
CAT
RUN AWAY
LIST
```

First enter and use the SAVE command to save an Applesoft program called /EXAMPLES/PROGRAMS/AWAY to be run by the EXEC program.

```
5  REM AWAY
10  PRINT "A WAY TO JOURNEY,"
20  PRINT "A WHALER JOE."
30  PRINT "AWEIGH THE ANCHOR,"
40  PRINT "AWAY WE GO."
```

Next write and save the following program, called MAKE.DOIT, which, when run, creates a text file called /EXAMPLES/PROGRAMS/DOIT.EXEC. The PRINT statements that begin with D$ are ProDOS commands; they are executed when the program is run. The other PRINT statements are written to the EXEC file, to be used later. Notice that ProDOS commands in an EXEC file, such as **RUN AWAY**, should *not* be preceded by a CONTROL-D.

```
5  REM MAKE.DOIT
10 D$ = CHR$ (4)                          D$ is (CONTROL)-(D).
20  PRINT D$;"PREFIX                      Set the prefix.
    /EXAMPLES/PROGRAMS"
30  PRINT D$;"OPEN DOIT.EXEC"             Prepare the file to be written to.
40  PRINT D$;"WRITE DOIT.EXEC"
50  PRINT "PREFIX                         Put these
    /EXAMPLES/PROGRAMS/"                  four commands
60  PRINT "CAT"                           into the
70  PRINT "RUN AWAY"                      EXEC file
80  PRINT "LIST"                          DOIT.
90  PRINT D$;"CLOSE DOIT.EXEC"            Close the file.
```

After you have MAKE.DOIT and AWAY both saved in the /EXAMPLES/PROGRAMS directory, type the command

```
RUN MAKE.DOIT
```

to create the sequential text file named DOIT.EXEC. To see the contents of DOIT.EXEC, you can once again use the program GET.TEXT.

Now type the command

```
EXEC DOIT.EXEC
```

to cause the commands in the file DOIT.EXEC to be executed one by one, just as if you were typing them—very quickly—from the keyboard. This EXEC program displays the files in the /EXAMPLES/PROGRAMS directory, which should include the files MAKE.DOIT, DOIT.EXEC, and AWAY; it displays the sentences printed out by AWAY; and finally it displays a listing of the program AWAY.

An All-Purpose EXEC Maker Program

Just as you can use GET.TEXT to look at an EXEC program, you can use the program MAKE.TEXT to create one. The only problem with this program is if you enter an erroneous line, there is no way to change it. Just make sure each line is correct before you press (RETURN)

Listing a BASIC Program to a File

A far more useful application of the EXEC command is to capture the listing of a BASIC program as a text file. Such a program can be used

- to edit a program using a word processor
- to place part of a program anywhere in another program
- to insert subroutines from a subroutine file into a program
- to connect two programs.

The following version of the CAPTURE program captures lines 2270 through 5130 of the program that is currently in memory in a text file named LISTING.EXEC. Replace the line numbers in line 5 of the program with the lines that you want to save, and replace the filename LISTING.EXEC with the name of the file in which you want the listing saved.

```
1  REM CAPTURE
2  D$ = CHR$ (4)                          D$ is (CONTROL)-(D).
3  PRINT D$; "OPEN LISTING.EXEC"          Prepare to write to the file LISTING.EXEC.
4  PRINT D$; "WRITE LISTING.EXEC"
5  LIST 2270,5130                         List the lines to the file.
6  PRINT D$; "CLOSE LISTING.EXEC"         Close the file.
7  END                                    End the program.
```

To use this program, you must already have a program in memory. Add these lines to those of the program in memory. If your program begins with line 10, you can add these lines with the same line numbers; alternately, you can change the line numbers so that they are all greater than the highest numbered line in your program.

If you placed CAPTURE at the beginning of your program, run it by typing

```
RUN
```

If CAPTURE is elsewhere in your program, type

```
RUN linenum
```

where linenum is the number of CAPTURE's first line.

Use EXEC to Combine Programs

Executing (EXEC) a file does not delete the program that is already in memory. Therefore, if you have a program in memory, and you EXEC a file that was created using CAPTURE, the lines from the text file program are added to the lines that are already in memory. This is a good way to combine programs or subroutines.

Machine Language to BASIC

Here's a program that reads (PEEKs) consecutive bytes of a machine-language program, and for each byte places a POKE statement into an EXEC program (POKER.EXEC). When the EXEC program is run, a BASIC program containing these POKE statements is entered into memory. You can use EXEC to place these lines into an existing BASIC program, or you can use EXEC to put them into memory when no other program is present and save it as a separate BASIC program.

```
5  REM  MACHINE LANGUAGE POKER
10 D$ = CHR$ (4)                    D$ is (CONTROL)-(D).
20  PRINT D$;"PREFIX                Set the prefix.
    /EXAMPLES/PROGRAMS"
30  PRINT D$;"OPEN POKER.EXEC"      Open, Close, and Delete
40  PRINT D$;"CLOSE POKER.EXEC"       deletes POKER.EXEC
50  PRINT D$;"DELETE POKER.EXEC"      even if it didn't exist.
60  PRINT D$;"OPEN POKER.EXEC"      Open POKER.EXEC.
70  PRINT D$;"WRITE POKER.EXEC"     Prepare to write to it.
80 LINENUMBER = 7000                First line number of program.
90  FOR PLACE = 768 TO 783          For each memory location,
100 :COUNTER = COUNTER + 1            increment counter.
110 : IF COUNTER = 10 THEN COUNTER  Put 10 POKES on each line.
    = 1
120 : IF COUNTER < > 1 THEN 150     For first POKE on a line,
130 : PRINT : PRINT LINENUMBER;       print the line number,
140 :LINENUMBER = LINENUMBER + 1      then increment line number.
150 : PRINT " POKE ";PLACE;",";     Poke a byte.
    PEEK (PLACE);" :";
160  NEXT PLACE                     And do next location.
170  PRINT                          New line for ProDOS command.
180  PRINT D$;"CLOSE POKER.EXEC"    Then close POKER.EXEC.
190  END                            All done.
```

To use this program, put the proper memory locations in line 90, change the line number in line 80 if you wish, and then run it. It creates the EXEC program POKER.EXEC (in the prefix directory). Next type

```
EXEC POKER.EXEC
```

and the lines containing the pokes are added to whatever BASIC lines are already in memory.

The EXEC Command

Use the EXEC command to take commands or data (all non-file input) from a sequential text file instead of the keyboard. It has the form

EXEC pn [,F#] [,S#] [,D#]

The F# option allows you to skip the first # lines of the text file.

The EXEC program currently in memory is not affected by the NEW command or the CLOSE command. An EXEC program cannot be stopped by (CONTROL)-(C). If an EXEC program uses the EXEC command to call another EXEC program, the second program replaces the first.

If an EXEC program runs a BASIC program, and the BASIC program contains a (non-file) INPUT statement, that input request is satisfied by data from the EXEC file. If you interrupt a running BASIC progam with (CONTROL)-(C), the remainder of the EXEC program usually is not executed.

Monitor commands cannot be executed from within EXEC programs.

The Options

pn	pn indicates the file containing the EXEC program.
[,F#]	# is the number of fields to skip at the beginning of the EXEC file.
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

Chapter 9

Binary Files

About This Chapter

This chapter describes the ProDOS commands that let you use binary programs and binary files on your disks. If you want only to run binary programs that are already on your disks, refer to the DASH (—) command in Chapter 4. This command moves any type of program from a disk file into memory and then start it running.

The commands in this chapter can be used

- to load, run, and save binary programs
- to use binary programs to read and write characters.

If you have an Apple IIc, refer to the Preface for "An Important Note for Apple IIc Owners."

This chapter also explains about system files and programs that are written in machine language. The end of this chapter has a section on ProDOS and the Monitor, and another section that explains how you can connect ProDOS to a clock/calendar chip so that files can be dated.

This Chapter's Commands

This chapter's commands are summarized below.

BRUN: Run a binary program from a file

The BRUN command: see also the DASH command, Chapter 4.

Use this command to transfer binary data from a binary disk file (type BIN) to a specified portion of memory; the program then executes automatically.

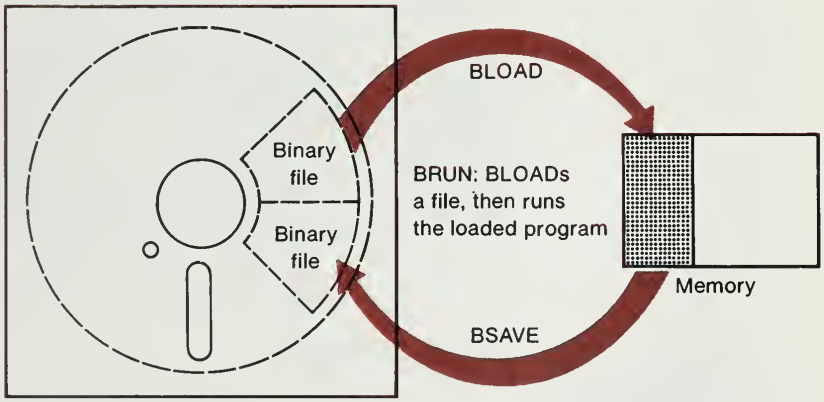
BLOAD: Read binary data from a file

Use this command to transfer binary data from any disk file to a specified portion of memory. Binary data is typically a machine-language program or a picture for one of the graphics screens.

BSAVE: Save binary data in a file

Use this command to transfer binary data from a specified portion of memory to any type of file. Data anywhere in the Apple II's memory can be transferred to a file.

Figure 9-1. BRUN, BLOAD, and BSAVE



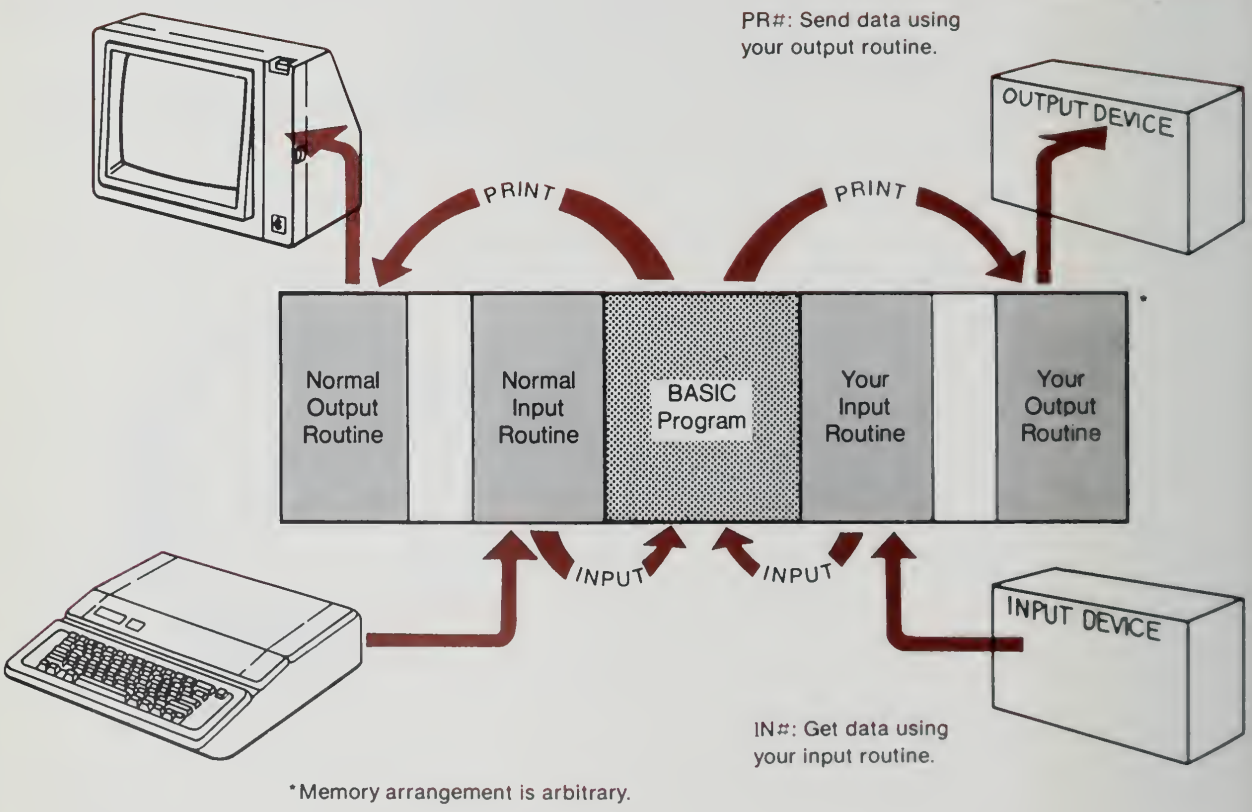
PR#: Use a binary program to print characters

You use this command most often to send output to a device in a slot. You can also use it so a binary program is used in place of the normal character output routine.

IN#: Use a binary program to read characters

You use this command most often to read characters from a device in a slot. You can also use it so a binary program is used in place of the normal character input routine.

Figure 9-2. PR# and IN#



Binary Files

ProDOS allows you to store on disk, and retrieve from disk, the information in your Apple II's memory. You have already seen the ProDOS commands RUN, LOAD, and SAVE: these commands deal with the contents of the BASIC program memory, interpreted as BASIC programs. The ProDOS commands discussed in the next three sections—BRUN, BLOAD, and BSAVE—perform similar functions, but they deal with the information in any part of the Apple II's memory, in its uninterpreted form.

The *B* before the commands BRUN, BLOAD, and BSAVE, stands for binary. Each command transfers binary information, zero-for-zero, and one-for-one, between memory and a file. From now on these commands are called the **binary commands**. The information that the binary commands transfer is often a machine-language program or a high-resolution picture from one of the Apple II's graphics screens, but it can be any information that is in memory or on a disk.

The two most common uses of the binary commands are: running the binary programs, and bringing binary images into memory for display. You can run a binary program using the DASH (—) command. If you have a file that contains a binary picture, you can move it to the graphics page from which it was transferred by the BSAVE command without having to understand the memory addresses involved. If you want to do this, use the example in the section on the BLOAD command.

The DASH command: see Chapter 4.

Binary Addresses

If you are going to be using the binary commands only to run machine-language programs that already exist, you don't have to understand the organization of the Apple II's memory. If, however, you want to save a graphics screen to a file, or if you want to work directly with the binary information in memory or in a file, you need to know a little about memory addresses.

Your Apple II's memory is a continuous sequence of memory locations, or bytes, each having an address. The address of the first memory location is 0 (written as $0000 in hexadecimal), the address of the second memory location is 1 ($0001), and so on. If your Apple II has 64K of memory, the address of the last memory location is 65535 ($FFFF).

The Memory Address Options: [,A#] [,E#] [,L#]

When you use a binary command to save binary information, you must give the address of the first memory location that contains information to be saved. The starting memory address is determined by the address option, A#.

You must also give the number of memory locations to be saved. You can do this using the length option, L#, which is the number of memory locations, or bytes, to be saved; or you can do it using the end address option, E#, the address of the last memory location to be saved.

For example, if you want to save high-resolution graphics screen 1 in a file, you must save the information that is in memory locations 8192 through 16383 ($2000 through $3FFF). Thus, to specify the start address, you can give the address option in decimal or hexadecimal

```
,A8192  or  ,A$2000
```

and you can give the address of the last byte to be transferred using the end address option, as in

```
,E16383  or  ,E$3FFF
```

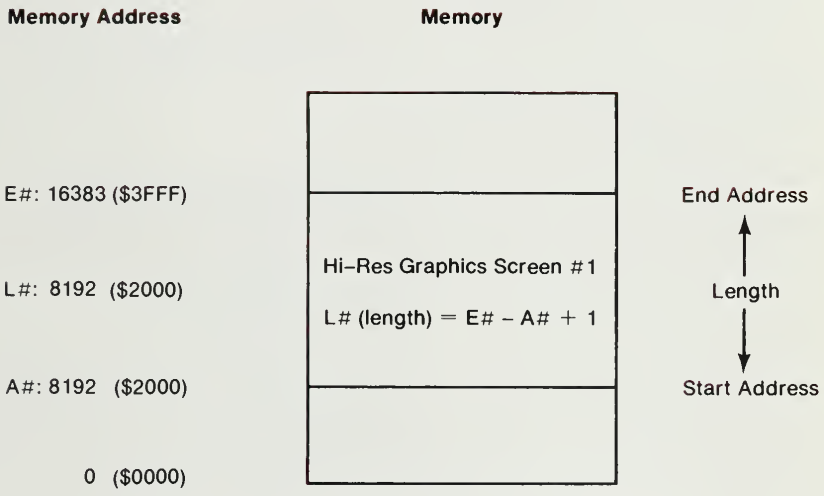
Alternately, if you like subtraction, you can use the length option to give the number of bytes to be transferred, as in

```
,L8192  or  ,L$2000
```

You can calculate the value for the length option by subtracting the start address from the end address and adding one.

The relationship between these three options for high-resolution graphics screen 1 is shown in Figure 9-3.

Figure 9-3. Memory Address Options A#, E#, and L#



The File Position Option: [,B#]

If you don't want to start the transfer of information with the first byte in a file, you can use the byte option, B#, to indicate the number of the first byte to be transferred.

For example, if you have two high-resolution pictures in a binary file, the first picture starts at byte 0 ($0000) in the file, and the second picture starts at byte 8192 ($2000) in the file. To address the second picture in the file, use the byte option

```
,B8192  or  ,B$2000
```

If you use the byte option, you must also use the length option, L#, or end option, E#, to specify the number of bytes to be transferred. If you don't, the entire file is transferred.

The BRUN Command

To run a binary program that is stored in a binary disk file (type BIN), use the command

BRUN pn [,A#] [,B#] [,L# | ,E#] [,S#] [,D#]

When ProDOS sees this command, it transfers the file indicated by pn into memory, as determined by the options, and then runs the program. If the A# option is not used, the program is placed in memory starting at the address from which it was transferred using BSAVE.

▲Warning

ProDOS cannot tell the difference between a binary program and binary data such as a picture file. If you give names to binary files that indicate their contents, such as adding .PIC to the end of all picture files, it is less likely that you will accidentally run a non-program file. If you ever run a non-program file, parts of ProDOS might change; if this happens, it is a good idea to restart ProDOS.

See Chapter 4 for the DASH command.

You can also run a binary program using the DASH command.

pn	pn must indicate a binary file (type BIN). If you use only this option, the entire contents of the binary file indicated by pn are placed into memory starting at the address from which they were transferred using BSAVE. You can see the address from which a file was transferred by BSAVE if you use the CATALOG command. This address is given, in hexadecimal, in the column labeled **SUBTYPE**, and it has an A in front of it.
[,A#]	# is the memory address into which the first byte of the program is to be transferred. It is not useful to use an address that is greater than the maximum memory address of your Apple II.
[,B#]	# is the number of the first byte in the file to be transferred. If you don't use this option, the first byte transferred is the first byte in the file, byte 0 ($0000).
[,L# | ,E#]	To load and run a portion of a file, use *one* of these options. L# is the number of bytes to be transferred; E# is the last memory address into which the program is to be transferred. If you include both of these options, the last in the list is used.
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

The BLOAD Command

To transfer binary information from a disk file to your Apple II's memory, use the command

BLOAD pn [,A#] [,B#] [,L# | ,E#] [,Ttype] [,S#] [,D#]

You can transfer information from any file type (the Ttype option), starting at any position in the file (the B# option), to any part of the Apple II's memory (the A#, L#, and E# options).

You can use the BLOAD command

- to transfer a machine-language program from a file to memory
- to move a picture from a file to a graphics screen
- to move the binary image of any type of file into memory.

Moving a binary image of any type of file into memory is discussed in the section "Using BLOAD With Non-Binary Files."

If you plan to write programs that use machine-language routines or high-resolution pictures, you need to take special precautions. The sections "High-Resolution Graphics With ProDOS" and "Installing Machine-Language Routines" address these issues.

The Options

If you give this command using only the filename option, the file indicated by pn must be a binary file (type BIN). The entire contents of this file are placed into memory starting at the address from which it was transferred by BSAVE.

pn — Unless you use the Ttype option, pn must indicate a binary file. If you use Ttype, pn can indicate any type of file. If you use the BLOAD command to transfer a non-binary file into memory, you *must* use the A# option.

[,A#] — # is the memory address into which the first byte of the binary data is to be transferred. You must use this option if you use BLOAD to load a non-binary file. If you use the BLOAD command to load a binary file without this option, the file is placed in memory starting at the address from which it was transferred by BSAVE.

[,B#]	# is the number of the first byte in the file to be transferred. If you don't use this option, the first byte transferred is the first byte in the file, byte 0 ($0000).
[,Ttype]	type is the three-letter abbreviation that indicates the type of file to be transferred. If no type is specified, the file must be of type BIN.
[,L# | ,E#]	To transfer a portion of a file into memory, use one of these options. L# is the number of bytes to be transferred. E# is the last memory address into which the data is to be transferred. You cannot use both these options in the same command.
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

For Example

With ProDOS started up, and the /EXAMPLES disk in drive 1, set the prefix to /EXAMPLES/DATA using the command

```
PREFIX /EXAMPLES/DATA
```

Now display high-resolution Page 1 and load in a picture by typing and running this program:

```
10 D$ = CHR$(4)                          D$ is CONTROL-D.
20  HGR                                  Display Page 1.
30  PRINT D$;"BLOAD PICTURE ,A8192       Load picture.
    ,E16383"
40  GET A$ : TEXT                        Restore text, after any key is pressed.
```

To clear the graphics page: see the section "High-Resolution Graphics With ProDOS."

To display text again (and to clear the graphics page), simply press any key.

Using BLOAD With Non-Binary Files

Token: an encoded element that represents a BASIC keyword.

When ProDOS places information in a file, the command that you use to save the information determines the format of the information in the file. For example, a BASIC program is saved as a set of BASIC tokens.

To see how a file is stored, and to work with the file in its uninterpreted form, use the BLOAD command with the Ttype option. For example, instead of manually changing every instance of a variable in a BASIC program, write a program that does it for you. Let the program use the BLOAD command to bring a BASIC program into memory, change all references to the variable, then save the program back to its BASIC file using BSAVE (with Ttype).

High-Resolution Graphics With ProDOS

Refer to the *Applesoft BASIC Programmer's Reference Manual* to see how to use HIMEM and LOMEM to protect the graghics page.

ProDOS does not ever prevent BASIC programs from overlapping the high-resolution graphics pages. To use high-resolution graphics Page 1 for graphics, use the HGR command; to use high-resolution Page 2 for graphics, use the HGR2 command.

▲Warning

On an Apple IIe or IIc, you cannot BLOAD data into high-resolution graphics Page 2 while 80-column text is being displayed.

When you finish using the graphics pages, use the TEXT command to return the screen to text mode. (The contents of the graphics page are not lost—they remain in memory.)

Note

If you want to go from high-resolution mode to text mode and back again without affecting the contents of the graphics pages, use the POKEs described in an appendix of the *Applesoft BASIC Programmer's Reference Manual*.

▲Warning

If your program halts—due to a STOP statement, an error, or a CONTROL-C typed from the keyboard—while you are using HGR or HGR2, do not attempt to continue the program using the CONT command. Type the CLOSE command, and run the program again.

Installing Machine-Language Routines

Refer to the *ProDOS Technical Reference Manual* for more details on machine language.

Because of the way ProDOS uses memory, it is difficult to predict which parts of memory are free to hold machine-language routines.

When ProDOS opens a file, it moves HIMEM down 1K and places a 1K file buffer where HIMEM used to be. It then marks that 1024-byte portion of memory as used in the system bit map.

To place a routine in memory, you must do the same thing: move HIMEM down by a multiple of 256 bytes, transfer the routine by using BLOAD, and then mark the used portions in the system bit map.

▲Warning

You must do this before any files are opened. This ensures that ProDOS places all file buffers below your routine, so your routine won't be closed instead of a file.

The BSAVE Command

To transfer binary information from your Apple II's memory to a disk file, use the command

BSAVE pn ,A# ,L# | ,E# [,B#] [,Ttype] [,S#] [,D#]

You can transfer information from any part of memory (the A#, L#, and E# options) to any type of file (the Ttype option), starting at any position in the file (the B# option).

You can use the BSAVE command

- to transfer a machine-language program from memory to a file
- to move a picture from a graphics screen to a file
- to move any portion of memory into any type of file.

The Options

When you use this command, you must use the pn, A#, and either the L# or the E# options.

pn	Unless you use the Ttype option, pn must indicate a binary file. If you use Ttype, pn can indicate any type of file.
,A#	You must use this option every time you use the BSAVE command. A# is the memory address from which the first byte of data is to be transferred.
,L# | ,E#	You must use one of these two options every time you use the BSAVE command. L# is the number of bytes of memory to be transferred. E# is the last memory address from which data is to be transferred.
[,B#]	B# indicates the first byte in the file to which data is to be transferred. If you don't use this option, the first byte is transferred to the first byte in the file, byte 0 ($0000).
[,Ttype]	type is the three-letter abbreviation that indicates the type of file to be transferred. If no type is specified, the file must be of type BIN.
[,S#]	The slot option has its usual meaning.
[,D#]	The drive option has its usual meaning.

For Example

This example loads a picture into graphics Page 1 and then saves it back into the same file. With ProDOS started up and the /EXAMPLES disk in drive 1, set the prefix using the command

```
PREFIX /EXAMPLES/DATA
```

Now, load the picture into high-resolution Page 1 and then save it using these commands

```
HGR                                  Turn on Page 1.
BLOAD PICTURE ,A16384 ,L8192         Load PICTURE.
BSAVE PICTURE ,A16384, L8192         Save PICTURE.
```

As a slightly more sophisticated example, here is how to move the files PRODOS and BASIC.SYSTEM from one disk to another using BLOAD and BSAVE. Assume that you are transferring the files from the volume /EXAMPLES to the volume /NEW.BOOT.

First type **CATALOG /EXAMPLES** to see how long the two files are. This is just part of the first two lines of a sample catalog:

```
/EXAMPLES

 NAME            TYPE    BLOCKS  ....   ENDFILE  SUBTYPE

*PRODOS          SYS         29            14336
*BASIC.SYSTEM    SYS         19  ....       9216
```

Here is how to perform the transfer:

```
BLOAD /EXAMPLES/PRODOS,TSYS,A8192
```
BLOAD the system program starting at 8192.

```
CREATE /NEW.BOOT/PRODOS,TSYS
```
Create a new system file.

```
BSAVE
/NEW.BOOT/PRODOS,TSYS,A8192,L14336
```
BSAVE the data from 8192 to the ENDFILE shown by CATALOG.

```
BLOAD
/EXAMPLES/BASIC.SYSTEM,TSYS,A8192
```
BLOAD the BASIC.SYSTEM file.

```
CREATE /NEW.BOOT/BASIC.SYSTEM,TSYS
```
Create a new system file.

```
BSAVE
/NEW.BOOT/BASIC.SYSTEM,TSYS,A8192
,L9216
```
BSAVE the data.

The start address, A8192, was chosen arbitrarily.

▲Warning Beware, however, this process destroys anything that was in the region of memory into which the BLOAD command placed the data.

The PR# and IN# Commands

Refer to the *Apple II* or *Apple IIe Reference Manual* for more details about accessory-card ROM space.

In addition to setting up the Apple II to do input or output with a slot, the PR# and IN# commands are also used to send characters to a machine-language program. The entire syntax of these commands is

PR# snum [,A#] or PR# A#
IN# snum or IN# A#

in which the option can either be the slot number (snum), the slot number followed by the address of the routine to be associated with that slot (A#), or just the address of a routine to be used (A#).

For example, if a character output routine is stored starting at memory location $300, you can use this command to activate the routine for subsequent output:

```
PR# A$300                    Output routine at location $300.
```

The first byte of the routine starting at location $300 must be a 6502 CLD instruction (216, $D8). When you want to stop using this routine, you can use this command to restore output to the console:

```
PR#0                         Restore output to console.
```

In addition, this same output routine can be associated with slot 2 using this command:

```
PR# 2,A$300
```

Subsequent references to slot 2 are actually directed to the routine at $300. To restore slot 2 to normal operation, use the command

```
PR# 2,A$C200
```

Use this form of the PR# command to remap physical slots from one slot number to another. For example, if you have printers in slots 1 and 2, and a program that expects the printer to be in slot 1, you can use the printer in slot 2 with your program by using the command

```
PR# 1,A$C200                 Assign slot 2 to slot 1.
```

before running the program.

▲Warning PR# and IN# are both ProDOS commands. When used from within a program, they must be preceded by a CONTROL-D. Otherwise, they are treated as Applesoft commands and cause ProDOS to be disconnected.

The Options

snum	snum can have any value from 0 to 7. If snum is 0, normal input or output to the console (keyboard and screen) is restored. If snum is from 1 to 7, the Apple II does subsequent input or output operations with the device in that slot.
A#	# is the address of the routine that you want to use as the character input or output routine. The first byte of the input or output routine must be a 6502 CLD instruction.

For Example

The first instruction of the input or output routine to be used must be a 6502 CLD instruction. As an example, put a two-line output routine starting at memory location $300. It consists of a jump to the normal Monitor output routine, located at memory address $FDF0.

With ProDOS started up, enter the Monitor with the command

```
CALL -151
```

and type

```
300: D8 4C F0 FD
```

```
0300: CLD
0301: JMP $FDF0
```

Now re-enter BASIC by pressing CONTROL-C and then pressing RETURN. Enter the ProDOS command

---

```
PR# $300
```

Output routine at $300.

and all subsequent output will be sent by the routine at location $300. This routine jumps into the normal character output routine, so characters are printed in their normal fashion. Type a few lines of BASIC, such as

```
PRINT "IT DOESN'T LOOK ANY DIFFERENT"
```

and you see that characters are indeed printed on the screen.

What PR# and IN# Really Do

This section explains the way the Apple II normally sends and receives characters; thus you will see how PR# and IN# work. The Apple II has two memory locations, named CSWH and CSWL, in which it stores the memory address of the routine that outputs characters. Together, these locations are called the **monitor output link**—they link the monitor to an output routine. It also has two memory locations, named KSWH and KSWL, in which it stores the memory address of the routine that inputs characters. These are called the **monitor input link**.

The monitor output link normally contains the address of the Apple II's standard output routine, COUT1; the monitor input link normally contains the address of the Apple II's standard input routine, KEYIN. These two routines send characters to the screen and read them from the keyboard, respectively. When you use PR# or IN# from BASIC, without ProDOS, the monitor links are set to indicate the ROM on the card in the indicated slot ($Cn00 for slot n). Thus, when the Apple II inputs or outputs a character, it calls the input or output routine in the card's ROM to perform the transfer.

While ProDOS is running, the monitor I/O links, instead of containing the addresses of the standard input and output routines, contain the addresses of the ProDOS input and output routines. ProDOS keeps the addresses of the standard input and output routines in the ProDOS input and output links. As you might expect, the ProDOS input and output links normally contain the addresses of the Apple II's standard input and output routines, KEYIN and COUT1.

When you use PR# or IN# with a slot number, ProDOS replaces the contents of the proper ProDOS link with the address of the ROM on the card in the indicated slot ($Cn00 for slot n). When you use PR# or IN# with an address, ProDOS simply places that address in the proper ProDOS link. When the Apple II tries to output or input a character, the monitor output or input links indicate the proper ProDOS routine, then the ProDOS routine does a two-stage transfer:

1. It moves the addresses of the current I/O routines from the ProDOS I/O links to the monitor I/O links. Then ProDOS calls the Apple II's normal I/O routines which use the current routines to perform the transfer.
2. ProDOS reconnects itself by placing the addresses of its I/O routines into the monitor I/O links.

Not only does ProDOS have input and output links for normal I/O, it also has them for each of the slots. When you use the PR# or IN# command with snum and A#, the specified address is placed in the links for that address.

ProDOS and the Monitor

If you like to play around with the Apple II's internals, you occasionally find yourself (intentionally or otherwise) in the Monitor. The Monitor is the program within the Apple II's read-only memory that controls many of the Apple II's vital functions.

First, to enter the Monitor from BASIC, type

```
CALL -151
```

and you see the monitor prompt

```
*
```

All ProDOS commands still work from within the Monitor. For example, type

```
CAT
```

and you see a normal catalog displayed on the screen. Likewise, the PR# command still starts up a disk from the Monitor. An error in a ProDOS command issued from the Monitor returns control to BASIC.

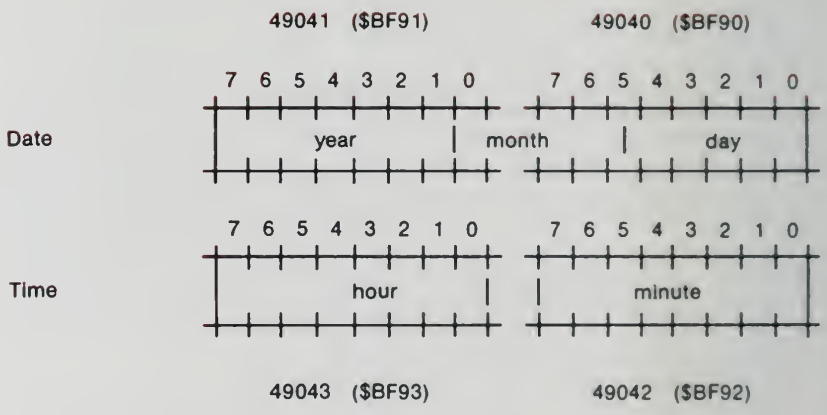
Re-enter BASIC by pressing CONTROL-C and then pressing RETURN.

Each time you update a file, ProDOS performs a JSR (jump subroutine) to memory location 48902 ($BF06). This is the entry into the DATETIME routine. If there is no DATETIME routine installed, there is an RTS in this location.

If, however, ProDOS recognizes a clock/calendar card in one of the slots, it sets up a routine and places a jump into the routine for you. If you want to use another type of clock/calendar card with ProDOS, you have to write your own routine, and place it in memory each time you start up ProDOS.

The routine must read the date and time from the card and place this data in bytes 49040 through 49043 ($BF90 through $BF93) using the following format:

Figure 9-4. ProDOS Date and Time Locations



A jump to the starting address of the routine must be stored in the entry to the DATETIME routine (48902) $BF06.

By the Way: The TIME program described in Appendix D does not change the time indicated by a clock/calendar card. It merely changes the system date and time locations described above.

System Programs

By now you have undoubtedly wondered why ProDOS is divided into two files—PRODOS and BASIC.SYSTEM—and what the relationship is between these two files.

The file PRODOS contains the most essential parts of ProDOS: routines that perform communication with disk drives in a compact and versatile way. The BASIC.SYSTEM file contains routines that let you communicate with disk drives through BASIC programs. When you use ProDOS BASIC, both these files are in memory and in use.

It is possible for other assembly-language programs to make use of the versatile routines in the PRODOS file without the overhead of having the BASIC.SYSTEM file in memory. Such programs are known as system programs.

You can recognize a system program by its file type, SYS, displayed by the CAT or CATALOG command. Every system program provides some way to switch from itself to another system program. From ProDOS BASIC, you run another system program using the DASH (–) command. From other system programs, you usually switch to another system program by using the Quit command.

Starting Up a System Program

When a disk starts up, the PRODOS file is first loaded into memory. Next a system program is loaded into memory. Then ProDOS scans the disk for the first file having the name XXX.SYSTEM (XXX can be combinations of letters and numbers that form a valid ProDOS filename). If it finds such a file, it loads it into memory and runs it. Otherwise it loads the first file of type SYS on the disk and runs it.

If there is no program of type SYS on the disk, an error message is displayed.

Appendix A

Summary of ProDOS

Features of ProDOS

Here is a list of some of the features of the ProDOS machine-language interface. These features are fully discussed in the *ProDOS Technical Reference Manual*. They are the basis upon which ProDOS is built.

- A directory-based filing system
- Up to 51 files in a volume directory; the number of files in other directories is limited only by space on the disk
- Up to 32 megabytes per volume
- Up to 16 megabytes per file
- 20 different file types (ten of them user-defined)
- Up to eight files can be open at a time
- A defined, usable machine-language interface
- A defined interrupt protocol
- File structures compatible with Apple III SOS
- Fast transfer rate—reads about 8K per second from Disk II
- Supports all Apple II disk devices

An understanding of these features is not essential to the summary of ProDOS that follows.

Filenames

A ProDOS filename is up to 15 characters long. It can contain uppercase and lowercase letters (A-Z), digits (0-9), and periods (.), and it must begin with a letter. Lowercase letters are automatically converted to uppercase.

A filename must be unique within its directory. Some examples are

ANGLOFILE
BALLOON
LETTER.TEXT

Pathnames

A ProDOS **pathname** is a series of filenames, each preceded by a slash (/). The first filename in a pathname is the name of a **volume directory**. Successive filenames indicate the path, from volume directory to the file, that ProDOS must follow to find that particular file. The maximum length for a pathname used in a command is 64 characters, including slashes.

Examples:

/EMPLOYEES/MARKETING/DIVISION.1
/SPORTS/FOOTBALL/THE.49ERS/QUARTERBACKS/MONTANA
/BIGDISK/RECORDS/MAY/JELLY.BEANS

Any command that requires you to name a file will accept a pathname or a partial pathname. A **partial pathname** is a portion of a pathname that doesn't begin with a slash. The maximum length of a partial pathname is 64 characters, including slashes.

These partial pathnames are all derived from the sample pathnames above:

MARKETING/DIVISION.1
DIVISION.1
THE.49ERS/QUARTERBACKS/MONTANA

When you use a partial pathname, ProDOS does one of two things. It usually adds the **prefix**, a pathname that indicates a directory, to the front of the partial pathname to form a complete pathname. But if the prefix is empty, or if you use either the slot option or the drive option (described in section "Syntax") in the command, the name of the volume specified by the slot and drive options is used instead of the prefix.

For the partial pathnames listed above to indicate valid files, the prefix should be set to /EMPLOYEES/, /EMPLOYEES/MARKETING/, and /SPORTS/FOOTBALL/, respectively. The maximum length for a prefix is 64 characters. You set the prefix using the PREFIX command.

Syntax

The syntax, or structure, of each ProDOS command is a command word followed by a list of options, as in

SAVE pn [,S#] [,D#]

The command word (SAVE in this example) is followed by a list of options. Unbracketed options must be included each time the command is used. Options in square brackets, [and] , are optional, and can be used in any order. Options separated by a vertical bar are alternates: use one or the other, not both (if both are entered, ProDOS uses the second in the list).

Uppercase letters and commas indicate characters that must be typed as shown; lowercase letters and the number symbol, #, stand for items that you supply.

Table A-1 is a summary of the command options. The next section contains a description of each of the options.

Table A-1. ProDOS Command Options

Name	Syntax	Minimum	Maximum	Examples	
Pathname	pn	†	†	/DISK/RECORDS/JAN	
Slot Number	,S#	1	7	,S1	,S3
Drive Number	,D#	1	2	,D1	,D2
Number of Fields	,F#	0 †	65535	,F2	,F10
Record Number	,R#	0	†	,R59	,R3982
Number of Bytes	,B#	0	†	,B2	,B7
Address in RAM	,A#	0	65535	,A512	,A4096
Length in Bytes	,L#	1	65535	,L10	,L16384
End Address in RAM	,E#	1	65535	,E776	,E32768
At Line Number	,@#	0	65535	,@10	,@322
Slot Number	snum	0	7	1	3
File Type	,Ttype	†	†	,TDIR	,TTXT

† See the description below.

Summary of the Options

This section describes the command options listed in Table A-1.

#	Decimal or hexadecimal integer. # can be replaced by a decimal integer, or it can be replaced by a hexadecimal number by preceding the hexadecimal digits with a dollar sign. The permitted values of # depend on the option.
pn	Pathname or partial pathname. See the section "Pathnames."
,S#	Slot number. # specifies an Apple II slot that contains a disk controller card. # initially defaults to the slot from which ProDOS was started up. It subsequently defaults to the last value specified for this parameter. # must be in the range 1 through 7. See Preface for IIc slot assignments. If # refers to a slot that does not contain a disk controller card, you will get the **NO DEVICE CONNECTED** error message.
,D#	Drive number (either 1 or 2). # initially defaults to one. It subsequently defaults to the latest value specified for this parameter. See Preface for IIc slot assignments. If ,S# is used without this option, ,D# defaults to one. If # refers to a drive that doesn't exist on the controller card in the indicated slot, you get the **NO DEVICE CONNECTED** error.

,F# Number of fields. Used with sequential and random-access text files. # specifies a field whose position in the file is # fields ahead of the current file position. # defaults to 0, which does not change the file position. Note: EXEC always sets the pointer to the start of the named file, so # is always relative to 0 when used with EXEC. Although # has a maximum value of 65535, if # specifies a position past the end of the record or the end of the file, the position pointer stops at the end of the record or file, and the **OUT OF DATA** error message is returned.

For DOS compatibility, both the options ,F# and ,R# indicate a number of fields when used with the EXEC or POSITION commands.

,R# Record number. Used with the READ and WRITE commands for random-access text files. # defaults to 0 after OPEN. Thereafter, it defaults to the last record specified. # points to an absolute record within a random-access file. The maximum record number is 16 megabytes divided by the file's record length, or 65535, whichever is smaller.

,B# Number of bytes. # defaults to 0. # indicates a position in a file whose position is # bytes ahead of the current position. For READ and WRITE it is evaluated after the ,F# option, and the maximum byte number is record length minus one. If it indicates a position past the end of record or file, the position is left at the end of the record or file, and the **END OF DATA** error message is returned.

For BRUN, BLOAD, and BSAVE, it is always used relative to the beginning of the file. If it indicates a position past the end of the file, the **RANGE ERROR** error message is returned.

,A#	Address in RAM. For BRUN, BLOAD, and BSAVE, # indicates a starting memory address for the transfer of binary information. If BLOAD does not specify this parameter, the value of A# defaults to that used when the binary file was transferred using the BSAVE command. For PR# and IN#, # specifies the memory address of a machine-language driver routine. # must be in the range 0 through 65535.
,L#	Length in bytes. # defaults to 1. In the OPEN and APPEND commands with random-access files, ,L# is required and specifies the record length in bytes. When used with the BRUN, BLOAD, and BSAVE commands, # specifies the number of bytes to be transferred between the Apple II's memory and a file. # must be in the range 0 through 65535. With the binary commands ,E# can be used instead of ,L#.
,E#	End address in RAM. This is an alternative to the ,L# option for BRUN, BLOAD, and BSAVE. # indicates the last memory address for the transfer of binary data. Either ,L# or ,E# is required for BSAVE. If neither is used with BRUN or BSAVE, bytes are transferred until the end of the file. # must be in the range 0 through 65535.

,@#	At line number. # indicates the number of the first program line to be executed by the RUN or CHAIN command. The default is the first line in the program. If there is no line with the number #, an error is returned. If the program is already in memory, the ,@# can be omitted.
snum	Slot number. snum is used with the IN# and PR# commands. It can have any value from 0 through 7. snum is the slot number of the device with which subsequent data (either input or output) is to be transferred. A snum of 0 specifies the Apple II's normal routines for I/O. See Preface for IIc slots.
,Ttype	File type. type is a three-letter abbreviation that indicates the type of file specified by the command. The possible values for type are given in Table A-2.

Table A-2. The File Type Abbreviations

Abbreviation	File Type
DIR	Directory
TXT	Text
BAS	Applesoft Program
VAR	Applesoft Variables
BIN	Binary
REL	Relocatable Code
$F# †	User Defined
SYS	ProDOS System File
SYS	ProDOS System Program

† # is an integer from 1 to 8.

As an example, the ProDOS command that has the syntax

READ pn [,R#] [,F#] [,B#]

can be interpreted as

```
READ /BIGDISK/MEMOS ,R100 ,F2
```

by the following process. The command word *READ* is in uppercase, and must be typed exactly as shown. The symbol for the pathname *pn* is in lowercase; it is replaced by the pathname **/BIGDISK/MEMOS**. The option ,R# becomes **,R100** indicating that data is to be read from record number 100 of the random-access text file **/BIGDISK/MEMOS**, which must already be open. The option ,F# is replaced by **,F2** indicating that the first two fields in record 100 are to be read and discarded before any data is taken from the record. The ,B# option is not used.

ProDOS Commands in Programs

You can give all ProDOS commands from within programs, and you can issue all except OPEN, WRITE, READ, APPEND, and POSITION from the keyboard. If any ProDOS command in a program is to be printed, it must be preceded by a printed (CONTROL)-(D), and the (CONTROL)-(D) must be the first character on the printed line.

Here is the most common way of printing a ProDOS command preceded by a CONTROL-D:

```
10 D$ = CHR$(4)                          D$ is set to CONTROL-D.
20 PRINT D$;"PRODOSCOMMAND"              Print the command preceded by
                                         CONTROL-D.
```

The WRITE, READ, and APPEND commands are terminated by the next ProDOS command given. If you want to terminate one of these commands without using a ProDOS command, you can use the null ProDOS command:

```
50 PRINT D$                              Null ProDOS command. D$ is CONTROL-D.
```

Filing Commands

This section contains a brief description of each ProDOS filing command.

CATALOG and CAT [pn] [,S#] [,D#]

Use the CATALOG and CAT commands in either immediate or deferred mode.

Examples:

```
CAT /DATABASE/PHOTOFILES                 List files in named directory.
CATALOG                                  List files in prefix directory.
CAT ,S6                                  List files in volume directory of slot 6, drive 1.
```

The CAT and CATALOG commands display a list of the files in the directory indicated by pn, S#, and D#. If pn is not used, a catalog of the prefix directory is displayed unless the prefix is empty or S# or D# is used. In these cases the volume name indicated by S# and D# is used instead of the prefix.

CAT displays a 40-column list containing the first five items explained below, while CATALOG displays an 80-column list with all eight items

For each file in the directory these commands display from left to right on the screen

- an asterisk if the file is locked (see the LOCK command)
- the file's name
- a three-letter abbreviation of the file's type (see Table A-2)
- the number of 512-byte blocks that the file occupies
- the date the file was last modified (Mo/Da/Yr Hr:Mn) (CAT displays Mo/Da/Yr only.)
- the date the file was created (Mo/Da/Yr Hr:Mn)
- the logical end of file (see section "ENDFILE (Maximum File Sizes)" in Chapter 3 and section "The End of File" in Chapter 7)
- the file's load address (in hexadecimal) if it is a binary file, or its record length (in decimal) if it is a random-access text file.

When you catalog a volume directory, the number of free blocks, used blocks, and total available blocks on that volume are displayed.

Possible Errors:

```
RANGE ERROR                NO DEVICE CONNECTED
PATH NOT FOUND             I/O ERROR
INVALID PARAMETER          NO BUFFERS AVAILABLE
FILE TYPE MISMATCH         SYNTAX ERROR
FILE BUSY
```

PREFIX [pn] [,S#] [,D#]

Use the PREFIX command in either immediate or deferred mode.

Examples:

```
PREFIX /PROFILE/WORKFILES/      Sets prefix to /PROFILE/WORKFILES/.
PREFIX ,D1 ,S6                  Sets prefix to name of volume in slot 6,
                                drive 1.
PREFIX /                        Make prefix empty.
PREFIX                          Display the prefix.
```

This command normally sets the value of the prefix, but if no options are used, the value of the prefix is displayed. If the command is used without options in a program, the next INPUT statement reads the value of the prefix. If a slash is used in place of pn, the prefix is set to empty; the volume specified by the default slot and drive is then used as prefix. Otherwise the standard rules for pn and the slot and drive options hold. The maximum length for the prefix is 64 characters, including slashes.

Possible Errors:

```
RANGE ERROR                NO DEVICE CONNECTED
PATH NOT FOUND             I/O ERROR
INVALID PARAMETER          FILE TYPE MISMATCH
SYNTAX ERROR
```

CREATE pn [,Ttype] [,S#] [,D#]

Use the CREATE command in either immediate or deferred mode.

Examples:

```
CREATE /PHONE/NEWDIR ,TDIR
CREATE FOTO ,D2 ,S6 ,TBIN
```

Creates a new directory.
Creates a binary file in volume directory, S6, D2.

Creates a file of the indicated type and name. Table A-2 shows the different file types. This command is primarily used for directory files. You must create a directory before saving a file in it.

Possible Errors:

```
RANGE ERROR                NO DEVICE CONNECTED
WRITE PROTECTED            PATH NOT FOUND
I/O ERROR                  DISK FULL
INVALID PARAMETER          SYNTAX ERROR
DIRECTORY FULL             DUPLICATE FILENAME
```

RENAME pn1,pn2 [,S#] [,D#]

Use the RENAME command in either immediate or deferred mode.

Example:

```
RENAME SEPERATE, SEPARATE, S4, D1
```

Changes the name of the file indicated by pn1, S#, and D# to the name indicated by pn2, S#, and D#. This command cannot move a file from one directory to another; it can only change the name of a file within its directory. You cannot rename a file that is locked.

Possible Errors:

RANGE ERROR	NO DEVICE CONNECTED
WRITE PROTECTED	PATH NOT FOUND
I/O ERROR	FILE LOCKED
INVALID PARAMETER	SYNTAX ERROR
DUPLICATE FILENAME	FILE BUSY

DELETE pn [,S#] [,D#]

Use the DELETE command in either immediate or deferred mode.

Example:

DELETE EXPLETIVE

Delete EXPLETIVE file.

Removes the file indicated by pn, S#, and D# from its directory. The file must not be open or locked. If the file is a directory file, it must be empty. You cannot delete a volume directory.

If a program tries to delete a nonexistent file, ProDOS returns the PATH NOT FOUND error message. To prevent this, open the file (which creates it if it doesn't yet exist), close it, then delete it.

Possible Errors:

RANGE ERROR	NO DEVICE CONNECTED
WRITE PROTECTED	PATH NOT FOUND
I/O ERROR	FILE LOCKED
INVALID PARAMETER	SYNTAX ERROR
FILE BUSY	

LOCK pn [,S#] [,D#]

Use the LOCK command in either immediate or deferred mode.

Example:

```
LOCK /GATES/PEARLY
```

This command protects the file indicated by pn, S#, and D# from being accidentally deleted, renamed, or changed. A locked file is indicated in the catalog by an asterisk (*).

Possible Errors:

```
RANGE ERROR                NO DEVICE CONNECTED
WRITE PROTECTED            PATH NOT FOUND
I/O ERROR                  INVALID PARAMETER
SYNTAX ERROR
```

UNLOCK pn [,S#] [,D#]

Use the UNLOCK command in either immediate or deferred mode.

Example:

```
UNLOCK RECIPES, S5
```

If the file indicated by pn, S#, and D# is locked, you must unlock it before you can alter, rename, or remove it.

Possible Errors:

```
RANGE ERROR                NO DEVICE CONNECTED
WRITE PROTECTED            PATH NOT FOUND
I/O ERROR                  INVALID PARAMETER
SYNTAX ERROR
```

VERIFY pn [,S#] [,D#]

Use the VERIFY command in either immediate or deferred mode.

Examples:

```
VERIFY /BOXES/LITTLE
```

PRINT CHR$ (4);"VERIFY /BOXES/LITTLE/

Use this command to verify that a particular file exists. For example, before creating a new file, use VERIFY to see whether a file with the same name exists already.

Possible Errors:

```
PATH NOT FOUND
SYNTAX ERROR
I/O ERROR
```

The message `PATH NOT FOUND` indicates that the file with the indicated name does not exist.

This section contains a brief description of the BASIC program commands.

—(DASH) pn [,S#] [,D#]

Use the DASH command in either immediate or deferred mode.

Example:

```
- /ANY/PROGRAM
```

Runs program in file /ANY/PROGRAM, regardless of program type.

This command, called the DASH command, can be used in place of RUN, BRUN, and EXEC; it is the only command that runs a system program. Thus, it can be used to run programs of types BAS, BIN, TXT, and SYS (XXX.SYSTEM files). Note that everything currently in memory is lost when a system program is invoked. (See the RUN, BRUN, and EXEC commands for more details.)

Possible Errors:

```
PROGRAM TOO LARGE          RANGE ERROR
NO DEVICE CONNECTED        PATH NOT FOUND
I/O ERROR                  INVALID PARAMETER
NO BUFFERS AVAILABLE       FILE TYPE MISMATCH
SYNTAX ERROR               FILE(S) STILL OPEN
```

RUN pn [,@#] [,S#] [,D#]

Use the RUN command in either immediate or deferred mode.

Examples:

```
RUN AMOK
```

Load BASIC program from file AMOK in prefix directory and run it.

```
RUN TWO.IN.ONE, @1000
```

Load BASIC program from file TWO.IN.ONE in prefix directory, and run it starting at line 1000.

Loads the Applesoft program in the file indicated by pn, S#, and D# (see the discussion of LOAD below), and then runs it. If @# is used, then the program starts running at the specified line; if that line is not found, the next highest line is run. Without @#, execution starts at the program's first line.

The RUN command, without any parameters, causes the BASIC program currently in memory to be run.

Possible Errors:

```
RANGE ERROR                NO DEVICE CONNECTED
PATH NOT FOUND             I/O ERROR
INVALID PARAMETER          FILE TYPE MISMATCH
PROGRAM TOO LARGE          SYNTAX ERROR
```

BASIC Errors:

```
UNDEF'D STATEMENT ERROR
```

LOAD pn [,S#] [,D#]

Use the LOAD command in either immediate or deferred mode.

Example:

```
LOAD DOW.JONES
```

Bring BASIC program in file DOW.JONES in prefix directory into memory.

This command tells ProDOS to search for the Applesoft program file (type BAS) with the name indicated by pn, S#, and D#. If there is such a file, its program is loaded into the Apple II's memory. The program can then be changed, listed, run, or saved. LOAD closes any open files (except EXEC files), and erases any BASIC program in memory before placing the new program in the Apple II's memory.

The instruction LOAD, without any parameters, attempts to load a program from cassette tape.

Possible Errors:

```
RANGE ERROR                NO DEVICE CONNECTED
PATH NOT FOUND             I/O ERROR
INVALID PARAMETER          FILE TYPE MISMATCH
PROGRAM TOO LARGE          SYNTAX ERROR
```

SAVE pn [,S#] [,D#]

Use the SAVE command in either immediate or deferred mode.

Example:

```
SAVE BABY.SEALS
```

Save current BASIC program in file BABY.SEALS.

If the indicated file does not exist, a file with the pathname indicated by pn, S#, and D# is created, and the current Applesoft program is stored in that file. If the file is locked, the `FILE LOCKED` error is returned.

▲Warning If a file with the indicated pathname already exists, its contents are replaced without warning by the current BASIC program. *Always lock all valuable files.*

The instruction SAVE, without any parameters, attempts to save a program onto cassette tape.

Possible Errors:

```
RANGE ERROR                NO DEVICE CONNECTED
WRITE PROTECTED            PATH NOT FOUND
I/O ERROR                  DISK FULL
FILE LOCKED                INVALID PARAMETER
FILE TYPE MISMATCH         SYNTAX ERROR
DIRECTORY FULL
```

Programming Commands

This section provides a brief description of each of the BASIC programming commands.

CHAIN pn [,@#] [,S#] [,D#]

Use the CHAIN command in either immediate or deferred mode.

Example:

```
CHAIN /EXAMPLES/PART.TWO
```

Used from within a BASIC program, it loads and runs the BASIC program specified by pn, S#, and D#, leaving the names and values of all the current variables in memory. This means that a program can operate on the results of the previous program, and it can leave data for any subsequently chained program.

If the @# option is used, execution of the indicated file begins at the specified line; if that line does not exist, the next highest line is run.

Possible Errors:

```
RANGE ERROR                NO DEVICE CONNECTED
PATH NOT FOUND             I/O ERROR
INVALID PARAMETER          FILE TYPE MISMATCH
PROGRAM TOO LARGE          SYNTAX ERROR
```

BASIC Errors:

```
UNDEF'D STATEMENT ERROR
```

STORE pn [,S#] [,D#]

Use the STORE command in either immediate or deferred mode.

Example:

```
STORE /GAMES/BINGO.VARS
```

Stores all BASIC variables.

This command packs all the currently defined BASIC variables, and writes them to the file (type VAR) indicated by pn, S#, and D#. These variables may be returned to memory using the RESTORE command.

Before storing Applesoft variables, ProDOS compacts the Applesoft string space. This may result in a delay of two to four seconds before the disk is actually accessed.

Possible Errors:

```
RANGE ERROR                NO DEVICE CONNECTED
WRITE PROTECTED            PATH NOT FOUND
I/O ERROR                  DISK FULL
FILE LOCKED                INVALID PARAMETER
FILE TYPE MISMATCH         SYNTAX ERROR
DIRECTORY FULL
```

RESTORE pn [,S#] [,D#]

Use the RESTORE command in either immediaate or deferred mode.

Example:

```
RESTORE /GAMES/BINGO.VARS
```

Load BASIC variables.

This command clears the current BASIC variables from memory, unpacks the variables stored in the variable file (type VAR) indicated by pn, S#, and D#, and puts the variables in the BASIC variable storage space in memory.

Possible Errors:

```
RANGE ERROR              NO DEVICE CONNECTED
PATH NOT FOUND           I/O ERROR
INVALID PARAMETER        FILE TYPE MISMATCH
PROGRAM TOO LARGE        SYNTAX ERROR
```

PR# snum | A# | snum ,A#

Use the PR# command in either immediate or deferred mode.

Examples:

```
PR# 1
PR# A$300
PR#1,A$C200
```

Send output to slot 1.
Send output using routine at $300.
Designate the ROM in slot 2 as the output routine for slot 1. Does not redirect output.

This command is used to send output to a slot; to send ouput to an address in memory; or to reassign the output address associated with a slot. It operates by changing the address of the current output routine (stored in memory locations $36 and $37). All subsequent non-file output is sent, a character at a time, by the routine at the specified address. If snum (a slot number) is used, the address of the current output routine is set to indicate the first byte of ROM on the card in that slot ($Csnum00). If A# is used, the address is changed to #, and the byte at this address must be a 6502 CLD (clear decimal) instruction.

Once the address of the output routine is changed, ProDOS performs a jump to this new address. The first portion of the code at that address normally performs initialization (such as starting up the disk in that slot). The code then resets the output routine address to indicate the true address of its output routine.

If both snum and A# are used, the specified address is assigned as the output address for that slot. It does not redirect output: a subsequent PR# snum must be used for this purpose.

See the description of PR# in Chapters 5 and 9 for more details; also see the *Apple II Reference Manual.*

Possible Errors:

```
RANGE ERROR                NO DEVICE CONNECTED
INVALID PARAMETER          SYNTAX ERROR
```

IN# snum | A#

Use the IN# command in either immediate or deferred mode.

Examples:

```
IN# 2
IN# A$300
```

Get input from slot 2.
Get input using routine at $300.

This command tells the Apple II to take its input from a slot or from an address in memory. It operates by changing the address of the current input routine (stored in memory locations $38 and $39). All subsequent non-file input is taken, a character at a time, by the routine at the specified address. If snum (a slot number) is used, the address of the current input routine is set to indicate the first byte of ROM on the card in that slot ($Csnum00). If A# is used, the address is changed to #, and the byte at this address must be a 6502 CLD (clear decimal) instruction.

Once the address of the input routine is changed, ProDOS performs a jump to this new address. The first portion of the code at that address normally performs initialization (such as starting up the disk in that slot). The code then resets the input routine address to indicate the true address of its input routine.

See the description of IN# in Chapters 5 and 9 for more details; also see the *Apple II Reference Manual.*

Possible Errors:

```
RANGE ERROR                NO DEVICE CONNECTED
INVALID PARAMETER          SYNTAX ERROR
```

FRE

Use the FRE command in either immediate or deferred mode.

Example:

```
PRINT CHR$ (4);"FRE"
```

This command removes any data remaining from former programs from the memory area used to store your program's string variables; that is, it cleans house.

Possible Error:

```
SYNTAX ERROR
```

BYE

Use the BYE command in either immediate or deferred mode.

Examples:

```
BYE
```

```
PRINT CHR$ (4);"BYE"
```

Use the BYE command to leave Applesoft BASIC and start another application program (either directly or through a program selector) without starting up the system (without "booting"). There are no options associated with this command.

Possible Error:

```
SYNTAX ERROR
```

Text File Commands

This section contains a brief description of each text file command.

OPEN pn [,L#] [,Ttype] [,S#] [,D#]

Use the OPEN command only in deferred mode.

Examples:

```
OPEN SESAME/SEED, D2          Open sequential file on drive 2 in default slot.
OPEN /X/RECORDS,L100          Open random-access file with record length
                              of 100.
OPEN DOOR,TDIR                Open a directory file.
```

This command allocates a memory buffer to the file indicated by pn, S#, and D#, and prepares the system to write or read from the beginning of the file. If the file did not previously exist, a text file is created. L# specifies the file's record length; if omitted, the record length defaults to the record length with which the file was opened, or to 1 for a new file.

Using Ttype you can open non-text files for reading and writing. Non-text files must be created before they can be opened. You must be careful when using this feature: the contents of non-text files can be difficult to deal with when using BASIC strings.

Up to eight files can be open at a time. The commands OPEN, CAT, CATALOG, and EXEC—and —(DASH) when you use it to execute (EXEC command) a file—all open a file. Only OPEN leaves the file open.

The memory buffer for an open file is 1024 bytes long. If there is not enough free memory for a file's buffer to be allocated, the file cannot be opened.

▲Warning A program must close all the files it opens. If it doesn't, data written to the file may be lost.

Possible Errors:

```
RANGE ERROR                   NO DEVICE CONNECTED
WRITE PROTECTED               PATH NOT FOUND
I/O ERROR                     DISK FULL
INVALID PARAMETER             NO BUFFERS AVAILABLE
FILE TYPE MISMATCH            NOT DIRECT COMMAND
SYNTAX ERROR                  DIRECTORY FULL
```

CLOSE [pn]

Use the CLOSE command in either immediate or deferred mode.

Examples:

```
CLOSE                    Close all open files.
CLOSE /P/NOSE            Close /P/NOSE.
```

The effect of CLOSE on an EXEC file: see the section "The EXEC Command." For comparison, see the FLUSH command.

The CLOSE command without options closes all open files (with the exception of EXEC files: see EXEC). If pn is used, only the specified file is closed. When a file is closed, any characters in the output part of the file buffer are written to that file, and its file buffer memory is released for other uses.

▲Warning

A program must close all files it opens. Failure to close an open file can result in loss of data. If a program terminates because of an error, because a CONTROL-C was pressed, or for any other reason, enter the CLOSE command from the keyboard before you do anything else.

Possible Errors:

```
I/O ERROR                    INVALID PARAMETER
SYNTAX ERROR
```

READ pn [,R#] [,F#] [,B#]

Use the READ command in deferred mode only.

Examples:

```
READ /EXAMPLES/HELPERS,R10   Prepare to read record 10 from
                             /EXAMPLES/HELPERS.
READ BOOK, F100              Skip 100 fields of the file BOOK and prepare
                             to read.
```

This command alters the current position and prepares input to be taken from the indicated file. If R# is used, the current file position is moved to the beginning of the specified record. If F# or B# is used, the current position is moved forward the specified number of fields and bytes.

Once this command is given, all characters asked for by INPUT or GET statements in the program are taken from the specified file starting at the file's current position. Each INPUT statement is ended by a carriage return character (ASCII code 13) or 224 bytes, whichever comes first. The READ command is terminated by the next ProDOS command, or by the null ProDOS command, that is, printing CONTROL-D.

▲Warning | Due to the limitations of BASIC strings, the reading of non-text files may not work as you expect it to.

For more details on READ, see the section "Reading From ProDOS Directories," Appendix D.

If you open and read a directory file, you get back strings that are identical in format to the lines returned by CATALOG.

Possible Errors:

```
RANGE ERROR                END OF DATA
I/O ERROR                  INVALID PARAMETER
NOT DIRECT COMMAND         SYNTAX ERROR
FILE NOT OPEN
```

WRITE pn [,R#] [,F#] [,B#]

Use the WRITE command only in deferred mode.

Examples:

```
WRITE /MYDISK/ADDRESS,R29      Prepare to write to start of record 29.
WRITE SOON                     Prepare to write to current position in file.
```

This command alters the current position, and prepares output to be sent to the indicated file. If R# is used, the current file position is moved to the beginning of the specified record. If F# or B# is used, the current position is moved forward the specified number of fields and bytes.

Once this command is given, all characters output by the program or by BASIC are placed in the specified file starting at the file's current position. The WRITE command is terminated by the next ProDOS command.

Although you can open directory files (type CAT), and read from them, you *cannot* write to them.

Possible Errors:

```
RANGE ERROR                WRITE PROTECTED
I/O ERROR                  DISK FULL
FILE LOCKED                INVALID PARAMETER
NOT DIRECT COMMAND         SYNTAX ERROR
FILE NOT OPEN
```

APPEND pn [,Ttype] [,L#] [,S#] [,D#]

Use the APPEND command only in deferred mode.

Example:

```
APPEND MORE.INFO
```

Prepare to write to end of file MORE.INFO .

This command opens the file specified by pn, S#, and D#, moves the current position to the end of the file, and issues a WRITE to that file. If L# is used (and is the same as the file's original record length), the current position is set to the beginning of the record immediately following the last record in the file.

Once this command is given, all characters output by the program or by BASIC are placed in the specified file starting at the file's current position. The WRITE part of the APPEND command is terminated by the next ProDOS command.

▲Warning Be sure that your program closes all appended files. Failure to do so may result in loss of data.

Possible Errors:

```
RANGE ERROR                NO DEVICE CONNECTED
WRITE PROTECTED            PATH NOT FOUND
I/O ERROR                  FILE LOCKED
INVALID PARAMETER          FILE TYPE MISMATCH
NO BUFFERS AVAILABLE       NOT DIRECT COMMAND
SYNTAX ERROR
```

FLUSH [pn]

Use the FLUSH command in either immediate or deferred mode.

Example:

```
FLUSH /WITH/DELIGHT
```

The FLUSH command without options causes all open files (with the exception of EXEC files: see EXEC) to be flushed. If pn is used, only the specified file is flushed. When a file is flushed, any characters in the output part of the file buffer are written to that file, and updated index and allocation buffers are copied to the file's directory.

▲Warning The FLUSH command is useful for preserving the integrity of the data on a disk. A program that may stop unexpectedly—whether due to power surges or little kids—should flush its buffers frequently. This way you can prevent much data from being lost.

Possible Errors:

```
RANGE ERROR                I/O ERROR
INVALID PARAMEER           SYNTAX ERROR
```

POSITION pn ,F# | ,R#

Use the POSITION command only in deferred mode.

Example:

`POSITION ADDRESS.DATA, F277` Read and discard 227 fields.

POSITION causes # fields to be read and discarded. For compatibility with DOS 3.3, the F# and R# options are functionally equivalent.

POSITION scans forward from the current position, character by character, until it encounters the #-th (RETURN) character following the current position. It then places the current position at the first byte following this (RETURN) character. If, in this search, it finds any byte in which no character has ever been stored (normally an end of record or end of file), the message `END OF DATA` is given.

Possible Errors:

```
RANGE ERROR                END OF DATA
I/O ERROR                  INVALID PARAMETER
NOT DIRECT COMMAND         SYNTAX ERROR
FILE NOT OPEN
```

The EXEC Command

The format and a description of the EXEC command are given in the section below.

EXEC pn [,F# | ,R#] [,S#] [,D#]

Use the EXEC command in either immediate or deferred mode.

Example:

```
EXEC PRIVILEGE,F3
```

Execute commands starting with the fourth field of the file PRIVILEGE.

This command causes the Apple II to take all (non-file) input from a sequential text file instead of from the keyboard. This allows you to use a text file containing BASIC or ProDOS commands, or input to a running program to control the operation of your Apple II.

Other uses of EXEC are explained in Chapter 8.

The file indicated by pn, S#, and D# must be a sequential text file (hereafter referred to as an EXEC file). ProDOS opens the EXEC file, reads and discards the number of fields specified by F# or R#, and then starts reading commands at that position. When the end of file is reached, the EXEC file is closed.

There can be only one EXEC command in effect at a time. If the EXEC file contains an EXEC command, the original EXEC file is closed and the new EXEC file is opened and executed. The CLOSE command, when issued from within an EXEC file, does not cause the EXEC file to close. If an EXEC file contains a RUN command, EXEC waits until the program ends, then the next command in the EXEC file is executed.

▲Warning

If a program is running while an EXEC file is open, an INPUT statement in the program reads its input from the EXEC file. Worse yet, if that response is an immediate-execution ProDOS command, the command is executed before the program continues.

Important

If you type CONTROL-C to stop an Applesoft program that is running while an EXEC file is still open, the remaining commands in the EXEC file are usually not executed.

For compatibility with DOS 3.3, the F# and R# options are functionally equivalent.

Possible Errors:

```
RANGE ERROR                NO DEVICE CONNECTED
PATH NOT FOUND             I/O ERROR
INVALID PARAMETER          NO BUFFERS AVAILABLE
FILE TYPE MISMATCH         SYNTAX ERROR
```

Binary Commands

The binary commands are briefly described in the next sections.

BRUN pn [,A#] [,B#] [,L# | ,E#] [,S#] [,D#]

Use the BRUN command in either immediate or deferred mode.

Example:

```
BRUN SUPER, A$C0A
```

The BRUN command loads the binary file (type BIN) indicated by pn, S#, and D# into the Apple II's memory as specified by A#, B#, and L# or E#. B# is the number of the first byte in the file to be loaded. A# is the first memory address into which data is to be loaded. L# is the number of bytes to be loaded, and E# is the last memory address into which data is to be loaded (either L# or E# should be used, not both). If A# is omitted, the file is loaded starting at the address from which it was saved. Once loaded, the file (which must be a machine-language program) is started by a machine-language jump (JMP) to address A#.

BASIC and ProDOS continue functioning if the machine-language program ends with a 6502 RTS instruction.

Possible Errors:

```
RANGE ERROR                NO DEVICE CONNECTED
PATH NOT FOUND             I/O ERROR
INVALID PARAMETER          FILE TYPE MISMATCH
SYNTAX ERROR
```

BSAVE pn ,A# ,L# | ,E# [,B#] [,Ttype] [,S#] [,D#]

Use the BSAVE command in either immediate or deferred mode

Examples:

```
BSAVE PICTURE, A16384, L8192
BSAVE PICTURE, A$4000, E$5FFF
```

The BSAVE command stores the contents of a segment of the Apple II's memory into a file with the name indicated by pn, S#, and D#, and the type indicated by Ttype. The default file type is binary (BIN). If the file does not yet exist, it is created. The segment is specified by the starting address A#, and either the number of bytes to be stored L# or the end address E#. B# specifies the starting file position.

The examples shown above both store a high-resolution picture from the second high-resolution picture area of memory. They have the same effect, but the second example uses hexadecimal notation and the E# option instead of the L# option.

Possible Errors:

```
RANGE ERROR                NO DEVICE CONNECTED
PATH NOT FOUND             WRITE PROTECTED
I/O ERROR                  DISK FULL
FILE LOCKED                INVALID PARAMETER
FILE TYPE MISMATCH         SYNTAX ERROR
DIRECTORY FULL
```

BLOAD pn [,A#] [,B#] [,L# | ,E#] [,Ttype] [,S#] [,D#]

Use the BLOAD command in either immediate or deferred mode.

Examples:

```
BLOAD PICTURE, A8192, L8192
BLOAD PICTURE, B$2000,A$4000,E$5FFF
```

The BLOAD command fills a segment of memory with data taken from the file with the name indicated by pn, S#, and D#, and with the type indicated by Ttype. The data is taken starting at file position B#, and is placed in memory starting at address A#. L# is the number of bytes transferred, and E# is the end address; one or the other, but not both, can be used. If A# is omitted, the first byte is placed at the address from which the file was originally saved (using BSAVE). If L# and E# are omitted, the last byte transferred is the last byte in the file.

For the examples, assume the file PICTURE has at least two high-resolution pictures in it, each 8192 bytes long. The first example shown above places the first 8192 bytes of PICTURE into the first high-resolution picture area, which starts at memory location 8192 (decimal). The second example moves the second picture, starting at byte position $2000 (8192) in the file, into the second high-resolution picture area, which starts at memory location $4000 (16384), and ends at memory location $5FFF (24575).

See the sections "High-Resolution Graphics With ProDOS" and "Installing Machine-Language Routines" in Chapter 9 for some important restrictions on memory usage.

Possible Errors:

```
RANGE ERROR                NO DEVICE CONNECTED
PATH NOT FOUND             I/O ERROR
INVALID PARAMETER          FILE TYPE MISMATCH
PROGRAM TOO LARGE          SYNTAX ERROR
```

Appendix B

DOS, ProDOS, and Applesoft

About This Appendix

This appendix summarizes the differences between DOS and ProDOS, by answering three questions:

- How do I tell what types of disks each of my programs can use?
- How do I tell which files can be converted from DOS to ProDOS and which can be converted from ProDOS to DOS?
- I already know about DOS. What's so different about ProDOS?

The final part of this appendix lists the effects of ProDOS upon some of the Applesoft commands.

The *ProDOS Technical Reference Manual* explains the many similarities between ProDOS and Apple III SOS (Sophisticated Operating System).

DOS Disks and ProDOS Disks

The first question, "How do I tell what types of disks each of my programs can use?" is a good one. It is not the program, but the way the program is stored on a disk, that determines the types of disk drives the program can use.

If a disk is formatted using the DOS command INIT, the programs on that disk use DOS, and DOS can use only Disk II drives or their equivalent.

If a disk is formatted by ProDOS, the programs on that disk use ProDOS, and ProDOS can use all disk drives made by Apple Computer, Inc. for the Apple II.

The question becomes: "How do I tell if a disk is DOS-formatted, ProDOS-formatted, or other?"

This problem exists only for 5¼-inch flexible disks: such a disk could be formatted for ProDOS, DOS 3.3, DOS 3.2.1 (or an earlier version), Apple II Pascal, or it could be unformatted. (If you have an Apple III, the disk could be SOS-formatted. In this case it is interchangeable with ProDOS disks.)

- If it is a 5¼-inch flexible disk, and the name on the label begins with a slash, it is probably ProDOS-formatted.
- If it is a 5¼-inch flexible disk, and the name on the label doesn't begin with a slash, follow the procedure given below.

By the Way: A Disk II disk is defined as a 5¼-inch single-sided flexible disk.

If you have the *System Utilities* disk, use the Identify and Catalog a Disk option. If that works, it is a ProDOS-formatted disk. If it doesn't work, try using the DOS-ProDOS Conversion Program. Set the direction to DOS —> ProDOS and then try to transfer files. If the program reads in a list of files, it is a DOS disk. If that doesn't work, the disk could be blank, copy-protected, or it could be an old version of DOS that used yet another method of storage.

If you can't list or catalog the files, try starting up the disk. If it doesn't start up, the program was probably stored on the disk using an old version of DOS. To use this disk, you need to use the DOS 3.3 disk labeled BASICS (which you get when you buy DOS) and follow the instructions in *The DOS Manual*. If the disk still doesn't start up, either the disk isn't formatted, or the information on the disk is damaged and is not readable.

If you aren't able to list a disk's files, but the disk starts up, the disk could be an Apple II Pascal disk, some other language, or it could be copy-protected. You probably have an instruction manual that tells about the filenames the program can use. If it mentions pathnames or prefixes, it is written using ProDOS; if it uses single filenames, it could be a DOS disk.

By the Way: Once you determine a disk's type, label the disk with its type for future reference. Always use a soft-tipped pen when writing on a disk's label.

Converting Files

ProDOS programs and data can use all types of disks made by Apple Computer, Inc. for Apple II computers, whereas DOS programs can use only Disk II disks. In addition, programs written using ProDOS read information from disks and write information to disks considerably faster than their DOS equivalents.

Following insrtructions in your *ProDOS User's Manual* or *System Utilities* manual, you can convert files from DOS format to ProDOS format, and back again. Table B-1 shows the correspondence between DOS and ProDOS files.

Table B-1. File Conversion

Contents of File	DOS Type		ProDOS Type
Text	T	<—>	TXT
Binary	B	<—>	BIN
Applesoft Program	A	<—>	BAS
Integer BASIC Program	I	<—>	INT
Relocatable Code File	R	<—>	REL
Other (ProDOS only)	B	<—	XXX

When converted, text and binary files are immediately usable by programs of the other type. Applesoft files usually have to be modified before they can be used. The following sections explain the changes you must make when modifying a DOS program.

The Differences Between DOS and ProDOS

There are three main areas of difference between DOS and ProDOS. First, ProDOS is an entirely different program from DOS: it uses different code and different parts of the Apple II's memory. Any DOS program that makes use of specific locations or routines in DOS will not work if converted. Likewise, programs that place assembly-language routines in memory may have to be changed so they don't overwrite parts of memory used by ProDOS. Details on the parts of memory used by ProDOS, and on the use of all the ProDOS routines, are given in the *ProDOS Technical Reference Manual*.

The second major difference is the conflicting filename conventions and file organizations used by DOS and ProDOS. It is likely that you will have to modify both when converting a program. In addition, ProDOS does not support volume numbers. If your program uses them, you will have to modify it to use volume names instead.

The third difference is the command structure. Some DOS commands are eliminated from ProDOS, other DOS commands have been improved, and several ProDOS commands are new. Of these changes, only the commands that have been eliminated will affect the programs that you are converting.

File Organization and Names

This section assumes that you are familiar with the organization and names of ProDOS files. If you are not, read Chapter 2 of this manual.

If a program refers to other files—for example, if it creates and uses a random-access text file, or if it chains to another program—then it is likely that you will need to change the way that the program names these files.

First, the filenames in a converted program must all be changed to ProDOS filenames: they can be no more than 15 characters long, consist only of letters, digits, and periods, and must begin with a letter.

Second, when the prefix is empty, ProDOS and DOS can use filenames in exactly the same way: each file has a filename, and the disk containing that file can be specified using the slot and drive options. Thus, if a converted program uses ProDOS filenames, it can work without further modification to the filenames, but only if the prefix is empty. The prefix is empty immediately after you start up ProDOS, and it also is empty after you use the command

The PREFIX command is explained in Chapter 3.

```
PREFIX /
```

However, it is preferable not to write or use programs that use files in this manner. Such programs place all files into a disk's volume directory. Because a disk's volume directory can hold no more than 51 files, it soon fills up if programs don't use directories of their own. You should revise each program so that it, and all its files, are stored in a separate directory.

DOS lets you assign a volume number to a disk when you initialize it. ProDOS does not support volume numbers. To modify a program that uses volume numbers, identify each disk by its volume name, not number.

DOS Commands That Went Away

Six DOS commands are not supported by ProDOS. They are:

FP	INIT	MON
INT	MAXFILES	NOMON

FP and INT

Because ProDOS supports only Applesoft BASIC, it has no need for commands that switch from one version of BASIC to another. If you use these commands, you get a SYNTAX ERROR.

INIT

Because of the different types of disks that must be initialized, it would take up too much memory space for ProDOS to have a built-in formatting command like INIT. Most application programs that involve the saving of data onto disks include a disk formatting routine. You can also use the *ProDOS User's Disk* or *System Utilities* disk to initialize disks.

Using INIT, you could assign any name to the greeting program on a disk; with ProDOS, a greeting program must be named STARTUP. However, ProDOS lets you use a BASIC, machine-language, or EXEC program as the STARTUP program; with DOS, only BASIC greetings were possible.

If ProDOS encounters the INIT command in a program, it gives you a SYNTAX ERROR.

MAXFILES

With DOS, the maximum number of files that could be open at once was three by default; this could be raised as high as 16 by using the MAXFILES command. With ProDOS, any program can have up to eight files open simultaneously. A 1024-byte file buffer is allocated to each file (or 512 bytes to a directory file) when it is opened.

See the *ProDOS Technical Reference Manual* for more details on open files.

The MAXFILES command is not supported by ProDOS, but it will not cause an error.

The maximum BASIC program size for a 64K Apple II is

$9600 - $400 * Maximum number of open files

MON and NOMON

With DOS, the commands MON and NOMON allowed you to display all disk commands, disk input, and disk output without printing them. These commands are not supported by ProDOS. MON has been completely removed. If your program uses MON, you will get a SYNTAX ERROR. NOMON is ignored by ProDOS, but will not cause an error.

Improved DOS Commands

Refer to the specific command summary in Appendix A for the new syntax.

Fourteen DOS commands received facelifts. All can be used in much the same manner as with DOS, so you don't have to change them when converting a program, but each has added features. The following sections describe the new capabilities of these commands.

The improved commands are:

APPEND	BLOAD	BRUN	BSAVE
CATALOG	CHAIN	CLOSE	IN#
PR#	OPEN	POSITION	READ
RUN	WRITE		

APPEND

The APPEND command has two new uses. You can now use it to append new data to any type of file. You can use it also to append data starting at the beginning of the record immediately following the last logical record in a random-access text file.

BLOAD

The BLOAD command has three enhancements. You can now use it to load the binary image of any type of file, not just binary files. With DOS you had to load an entire binary file into memory. With ProDOS you can load any portion of a file. In addition, you have the option of specifying the number of bytes to be transferred as a start address and an end address in memory, or as a start address and the number of bytes to be transferred.

BRUN

As with BLOAD, you can load any portion of a binary file into memory and run it. The number of bytes can be specified using a start and an end address, or as a start address and the number of bytes to be transferred.

BSAVE

You can now transfer (using BSAVE) information stored in memory into any type of file. The number of bytes to be saved can be specified using a start address and an end address, or as a start address and the number of bytes to be transferred.

CATALOG

You can now abbreviate the CATALOG command as CAT. CATALOG shows you an 80-column display of file information, and CAT shows you a 40-column display of information. Both show you the contents of a single directory; thus you must specify the name of the directory whose contents interest you. If you omit a filename, you see the contents of the prefix directory.

In addition, CATALOG now displays information about the file's logical end of file, the file's record length (random-access text files); the file's last load address (binary files); and the dates when the file was created and last modified.

CHAIN

The CHAIN command now works for Applesoft programs (with DOS it did not). In addition, one program can chain to any line of another program, not just to the beginning of a program as before.

CLOSE

Now you *must* close a file from within a program. Failure to do so can result in loss of data.

IN# and PR#

You can now use these commands to designate machine-language routines stored in memory as the character input and output routines. You can also use them to set the address of a slot's I/O routines.

OPEN

You can now use the OPEN command to open any type of file for access. File buffers are now allocated when the file is opened, instead of in response to the MAXFILES command, as with DOS.

POSITION

The POSITION command now uses either the F# option or the R# option to read and discard the specified number (#) of fields from a file. The F# option is consistent with the ProDOS definition of fields; the R# option is for DOS compatibility.

The ProDOS versions of READ and WRITE allow you to specify the number of fields and bytes to be read and discarded. Thus, the POSITION command is not required by ProDOS; it is retained for compatibility with DOS.

READ

With DOS, the READ command allowed you to use the B# option to position forward a number of bytes before performing a read. The ProDOS version of READ allows you to use the F# and B# options to position forward a number of fields and bytes. This makes the POSITION command unnecessary for READ (and also for WRITE).

RUN

The ProDOS version of the RUN command allows you to specify the line number at which the program is to start running.

WRITE

The ProDOS version of the WRITE command allows you to use the F# and B# options to position forward a number of fields and bytes.

New ProDOS Commands

In addition to the usual DOS commands, ProDOS supports several new commands. They are:

CAT	CREATE	FLUSH	PREFIX
STORE	RESTORE	—(DASH)	FRE
VERIFY	BYE		

The following sections give brief summaries of each of these new commands.

Appendix A contains summaries of these commands.

CAT

CAT displays 40 columns of directory file information, while CATALOG displays 80 columns. Both commands display filenames, their types, lengths, and last modified dates. CATALOG additionally displays the date the file was created, each file's logical end of file, and additional storage information (record length for random-access text files, and last load address for binary files).

CREATE

The CREATE command allows you to create a file of any type, but it is primarily used to create directory files. BASIC (type BAS), text (type TXT), and binary (type BIN) files are automatically created by the SAVE, OPEN, and BSAVE commands, respectively. Text files can also be created by the APPEND command. Variable files (type VAR) are created by the STORE command.

Note | You do not have to specify the size of a created file. Additional blocks are added to a file as they are needed.

FLUSH

The FLUSH command causes all data that may be temporarily stored in a file's buffer to be written to the file. Using the FLUSH command after every statement that prints data to a file ensures that no data will be lost if the program is accidentally stopped. It also slows down a program significantly.

PREFIX

The PREFIX command allows you to set the name of the directory that contains the files with which you are working. With the prefix set, all files you name are assumed to be in that directory.

If the prefix is empty, files are assumed to be in the main directory of the disk in the last referenced slot and drive. In this case, ProDOS filenames work exactly like DOS filenames.

STORE and RESTORE

The STORE command places the names and values of all the variables currently defined by a BASIC program into a variable file (type VAR). The RESTORE command adds the contents of a variable file to the variables that are currently in memory.

—(DASH)

This command, consisting of a single character, is called the DASH command. It is a generic RUN command, allowing you to run a BASIC, binary, EXEC, or system program. It does not let you use any of the specific options afforded by the RUN, BRUN, or EXEC commands.

FRE

FRE command: see the section "Changes to Applesoft."

The FRE command lets you use the ProDOS fast housekeeping routines.

VERIFY

Use this command to verify that a particular file exists. For example, before creating a new file, use VERIFY to see whether a file with the same name already exists.

BYE

Use this command to leave Applesoft BASIC and start another application program (either directly or through a program selector) without starting up the system ("booting").

Changes to Applesoft

In order to keep a congenial working relationship between ProDOS and Applesoft, it is necessary for ProDOS to intercept and perform some of the commands usually performed by Applesoft. The following sections explain the new enhancements or restrictions upon these ten commands:

HIMEM	HGR	HGR2	TEXT
INPUT	IN#	PR#	TRACE
NOTRACE	FRE		

HIMEM

Using HIMEM to place machine-language routines safely in memory: see the *ProDOS Technical Reference Manual.*

Each time a file is opened, ProDOS uses the HIMEM setting to determine where it should place the file's I/O buffer. Because ProDOS manages memory in 256-byte chunks, you must always make sure that HIMEM indicates a 256-byte ($100) boundary in memory.

HGR, HGR2, and TEXT

Refer to the section "High-Resolution Graphics With ProDOS" in Chapter 9.

Because the Apple II's two high-resolution pages take up a considerable portion of the Apple II's memory, ProDOS normally uses them as Applesoft program memory. If, however, you use the HGR or HGR2 command (or both), the Applesoft program (if any) is cleared out of the corresponding graphics page. The graphics pages remain available for graphics until the TEXT command is issued.

INPUT

The Applesoft INPUT command has been made more useful. This command always reads an entire line of text from a file. As before, multiple variables in an INPUT statement are assigned strings of characters that are separated by commas in the input string. When you use ProDOS, the last variable in the INPUT list is assigned all the remaining characters in the line, including commas and colons. This means that you can now use a single INPUT statement, such as

```
10  INPUT XX$
```

to read in any arbitrary string of characters.

IN# and PR#

If you use one of these commands from immediate mode, it is a ProDOS command. Likewise, if you use one from within a program, preceded by CONTROL-D, it is also a ProDOS command. If you use IN# or PR# from within a program without a leading CONTROL-D, it is an Applesoft command, and would cause ProDOS to become disconnected if it were executed. Thus, ProDOS intercepts these Applesoft commands and ignores them.

By the Way: If you find that a PR# or IN# command from within a program is not having the proper effect, you probably forgot the CONTROL-D.

TRACE and NOTRACE

These Applesoft commands did not work with DOS. They now have their normal effect, described in the *Applesoft BASIC Programmer's Reference Manual*, on ProDOS commands as well as Applesoft commands.

FRE

If you use the FRE command within a program preceded by a CONTROL-D, it is a ProDOS command. It is an Applesoft command if you use it in a program without the leading CONTROL-D. In this case, housekeeping takes place using the slow Applesoft routines.

If you use the ProDOS FRE command followed by the Applesoft command `PRINT FRE (X)`, the amount of free memory will be displayed without a wait for Applesoft housekeeping.

Appendix C

Error Messages

When ProDOS detects an error caused by one of its commands, it normally stops the program that is running and displays a message describing the error. The messages are in addition to the usual messages generated by Applesoft. The source of an error message is indicated by the character that precedes the message. Table C-1 illustrates these characters.

Table C-1. Error Message Formats

Applesoft Message	ProDOS Message
?SYNTAX ERROR	SYNTAX ERROR

If you get a ProDOS error message when you are using the Monitor, the system is reset to BASIC before the message is displayed.

Handling Errors From Applesoft

Refer to Chapter 5 of this manual and to the *Applesoft BASIC Programmer's Reference Manual* for more details about ONERR GOTO.

Using Applesoft's ONERR GOTO command, you can write Applesoft error-handling routines to correct ProDOS and Applesoft errors that would normally interrupt your program.

When a ProDOS or Applesoft error occurs following an ONERR GOTO command in an Applesoft program, a code number for the type of error is stored in decimal memory location 222. The statement

```
E = PEEK (222)
```

sets the value of `E` to the code of the offending error. The number of the Applesoft program line being executed at the time of the error can be found in decimal locations 218 and 219.

The statement

```
L = PEEK (218) + PEEK (219) * 256
```

sets the value of `L` to that line number.

The ProDOS error messages, their codes, and the most common cause for each are described in Table C-2. Table C-3 shows which error messages are caused by each of the ProDOS commands. The ProDOS error messages are discussed in greater detail later in this appendix The Applesoft error codes and their corresponding messages are shown in Table C-4.

Table C-2. ProDOS Error Codes

Code	ProDOS Message	Most Common Cause
2	RANGE ERROR	Command option too small or large.
3	NO DEVICE CONNECTED	No device found in specified slot.
4	WRITE PROTECTED	Write-protect tab on disk.
5	END OF DATA	Read beyond end of file or record.
6	PATH NOT FOUND	No file with indicated pathname.
7	PATH NOT FOUND	No file with indicated pathname.
8	I/O ERROR	Door open, or disk not formatted.
9	DISK FULL	Too many files on a disk.
10	FILE LOCKED	Attempt to write to a locked file.
11	INVALID PARAMETER	Option inappropriate for command.
12	NO BUFFERS AVAILABLE	Memory full, file can't be opened.
13	FILE TYPE MISMATCH	Disk file wrong type for command.
14	PROGRAM TOO LARGE	Apple II's memory too small. (CHAIN).
15	NOT DIRECT COMMAND	Command must be in a program.
16	SYNTAX ERROR	Bad filename, option, or comma.
17	DIRECTORY FULL	Volume directory has 51 files.
18	FILE NOT OPEN	Attempt to access a closed file.
19	DUPLICATE FILENAME	RENAME, CREATE name already used.
20	FILE BUSY	File already open.
21	FILE(S) STILL OPEN	Last program didn't close file(s).

Table C-3. Errors by ProDOS Command

	02 RANGE ERROR	03 NO DEVICE CONNECTED	04 WRITE PROTECTED	05 END OF DATA	06 PATH NOT FOUND	07 PATH NOT FOUND	08 I/O ERROR	09 DISK FULL	10 FILE LOCKED	11 INVALID OPTION	12 NO BUFFERS AVAILABLE	13 FILE TYPE MISMATCH	14 PROGRAM TOO LARGE	15 NOT DIRECT COMMAND	16 SYNTAX ERROR	17 DIRECTORY FULL	18 FILE NOT OPEN	19 DUPLICATE FILENAME	20 FILE BUSY	21 FILE(S) STILL OPEN	UNDEF'D STATEMENT ERROR	BAD BRANCH ERROR
APPEND	x	x	x		x	x	x		x	x	x	x		x	x							
BLOAD	x	x			x	x	x			x		x	x		x							
BRUN	x	x			x	x	x			x		x			x							
BSAVE	x	x	x		x	x	x	x	x	x		x			x	x						
CAT	x	x			x	x	x			x	x	x			x				x			
CATALOG	x	x			x	x	x			x	x	x			x				x			
CHAIN	x	x			x	x	x			x		x	x		x						x	x
CLOSE							x			x					x							
CREATE	x	x	x		x	x	x	x		x					x	x		x				
DELETE	x	x	x		x	x	x		x	x					x				x			
EXEC	x	x			x	x	x			x	x	x			x							
FLUSH	x						x			x					x							
FP															x					x		
FRE															x							
INT															x					x		
IN#	x	x								x					x							
LOAD	x	x			x	x	x			x		x	x		x							
LOCK	x	x	x		x	x	x			x					x							
OPEN	x	x	x		x	x	x	x		x	x	x		x	x	x						
POSITION	x			x			x			x				x	x		x					
PREFIX	x	x			x	x	x			x		x			x							
PR#	x	x								x					x							
READ	x			x			x			x				x	x		x					
RENAME	x	x	x		x	x	x		x	x					x			x	x			
RESTORE	x	x			x	x	x			x		x	x		x							
RUN	x	x			x	x	x			x		x	x		x						x	x
SAVE	x	x	x		x	x	x	x	x	x		x			x	x						
STORE	x	x	x		x	x	x	x	x	x		x			x	x						
UNLOCK	x	x	x		x	x	x			x					x							
WRITE	x		x				x	x	x	x				x	x		x					
—	x	x			x	x	x			x	x	x	x		x				x	x	x	

Table C-4. Applesoft Error Codes

Code	Error Message or Description
0	NEXT WITHOUT FOR
16	SYNTAX ERROR
22	RETURN WITHOUT GOSUB
42	OUT OF DATA
53	ILLEGAL QUANTITY
69	OVERFLOW
77	OUT OF MEMORY
90	UNDEF'D STATEMENT
107	BAD SUBSCRIPT
120	REDIM'D ARRAY
133	DIVISION BY ZERO
163	TYPE MISMATCH
176	STRING TOO LONG
191	FORMULA TOO COMPLEX
224	UNDEF'D FUNCTION
254	Bad response to an INPUT statement.
255	CONTROL-C interrupt attempted.

Refer to the *Applesoft BASIC Programmer's Reference Manual* for more information about the Applesoft error codes.

Discussion of ProDOS Errors

The following sections list the ProDOS errors, their probable causes, and possible cures.

RANGE ERROR (Code 2)

Occurs when the value of an ProDOS command option is too large or too small. Table A-1 shows the maximum and minimum values for each option.

Note

The use of values outside the indicated ranges does not always cause the RANGE ERROR message. Any ProDOS command option that is less than 0 or greater than 65535 causes the SYNTAX ERROR message, not the RANGE ERROR message.

NO DEVICE CONNECTED (Code 3)

Occurs when you specify a slot that doesn't contain a card; a slot that contains a card not connected to its device; or if there is no disk in the drive (some drives only).

If you get this error when using a device for the first time, go through the device's installation instructions again. If you have used the device before and you get this error:

1. You might have specified the wrong slot; try the command again.
2. Turn off your Apple II, open it, and gently rock the device's card back and forth until it is firmly seated. Close the Apple II, start up ProDOS, and try again.
3. If the problem persists, consult your dealer.

WRITE PROTECTED (Code 4)

Occurs when ProDOS attempts to store information on a disk, and the disk drive does not detect a write-enable notch or cutout on the disk's outer case. These are the most likely causes:

- There is an adhesive label placed over the disk's write-enable notch to prevent the writing or deletion of information on the disk. Remove this label, and the disk is no longer protected.
- There is no write-enable notch on the disk. This is true of the original copy of the /EXAMPLES disk, for example. If you are sure you no longer need the original disk, you can cut a notch in the disk's case yourself. Use another disk's notch as a template.

END OF DATA (Code 5)

Occurs when you try to retrieve information from a portion of a text file where no information has ever been stored. Any byte beyond the last field in a sequential text file, or beyond the last field of any record of a random-access text file contains the value 0. Zero is the ASCII code for a null character, a *nothing*; any command that causes the retrieval of this character results in the `END OF DATA` message. The message usually occurs after an INPUT or a GET command.

This message can arise in several different ways:

- Too many successive INPUTs or an INPUT with too many variables. Each INPUT statement causes at least one additional adjacent field to be read into the Apple II. Each INPUT variable causes one additional adjacent element to be read into the Apple II.
- Too many successive GETs. Each GET reads one additional adjacent byte or character into the Apple II.
- The B# (Byte) option in a READ or POSITION command is too large. In sequential files, this option must not specify a byte beyond the last (RETURN) character in the file. In random-access files, the B# option must not specify a byte beyond the last (RETURN) character in the currently selected record. Remember, the first byte in a file or a record is byte 0.
- The F# (Field) option in a READ or POSITION command is too large. In sequential files, this option must not specify a field beyond the last existing field in the file. In random-access files, POSITION's F# option must not specify a field beyond the last existing field in the currently selected record.

  READ and POSITION scan forward through the contents of the file, byte by byte, looking for the F#-th (RETURN) character. If either encounters a 0 byte (the null character) before finding the required (RETURN) character, the **END OF DATA** message is given immediately: it is not necessary actually to INPUT or GET the null character.

- The F# (Field) option in an EXEC command is too large. This option can specify the first field beyond the last existing field in a file, but attempting to specify the second field beyond the file's end causes the **END OF DATA** message. Remember, R0 specifies the first field in a file.
- The R# (Record) option in a READ command specified a random-access file record in which nothing is yet stored. Before you can READ from a particular record in a random-access file, you must first WRITE some information into that record. R0 is the file's first record, and so on.

ProDOS uses the OPEN command's L# option for calculating where the R#-th record begins, so the OPEN preceding READ must use the same L# option value as the OPEN that preceded WRITE for that file. If no L# option is specified, the L# with which the file was originally opened is used.

PATH NOT FOUND (Code 6 or 7)

Occurs when a ProDOS command specifies a valid pathname that does not indicate an existing file, or when it specifies an invalid pathname.

This message may arise in various ways:

- You accidentally misspelled an element of the pathname.
- You used a partial pathname that doesn't apply to the current prefix.
- You used a partial pathname, and the disk indicated by the prefix is no longer on line.
- The specified file does not yet exist.

I/O ERROR (Code 8)

Occurs after an unsuccessful attempt to store data or retrieve data (ProDOS tries 96 times, then gives up). This message can occur in the following ways:

- The selected or default drive's door is open. Close the door of the disk drive.
- There is no disk in the disk drive indicated by S# and D#. Put a disk into the drive and close the drive door.
- The disk in the selected or default disk drive is not formatted. Use the ProDOS Filer to format the disk.
- The disk is incorrectly seated in the disk drive. Open the drive door, pull the disk out, put it back in, close the door, and try again.
- The ProDOS command's D# (Drive) option specified a non-existent disk drive. The default drive is now the non-existent drive. Just specify the correct D# option with the next ProDOS command to reset the default.
- A ProDOS command's S# (Slot) option specified a slot that does not contain a disk controller card, or the snum option of PR# or IN# specified a slot that contains no card.

The default value of S# now indicates a slot that doesn't exist. First, you must get a prompt back, then you must reset S#. To reset the slot correctly:

1. Press (CONTROL)-(RESET).
2. If you see a Monitor prompt (*), press (CONTROL)-(C) (RETURN)
3. Type `CATALOG S#, D#` This time use a valid slot number.

DISK FULL (Code 9)

Occurs when ProDOS attempts to store information on a disk and finds that no more storage space is available on that disk. The number of free blocks on a disk is indicated when you display the catalog of the disk's volume directory. If you receive the **DISK FULL** message, rest assured that all files are closed, and that ProDOS saved for you all it could (leaving you with some portion of your file not on the disk). If you receive this message while saving a file called STUFF, the first thing you should do is

```
DELETE STUFF
```

and then save your program on another disk that has unused room.

FILE LOCKED (Code 10)

Occurs when you try to APPEND, BSAVE, DELETE, RENAME, SAVE, STORE, or WRITE a locked file. Check the CATALOG display: the names of locked files are preceded by an asterisk (*). Files are locked to prevent their being accidentally overwritten. Use another disk or unlock the desired file.

INVALID PARAMETER (Code 11)

Refer to Appendix A to see what options go with which commands.

Occurs when you use an option that is either non-existent or that is inappropriate for the given command.

NO BUFFERS AVAILABLE (Code 12)

When a file is opened by the APPEND, CAT, CATALOG, EXEC, OPEN, or —(DASH) command, a 1K buffer in memory is assigned for the temporary storage of data and file information. There can be a maximum of eight files open at a time. This error can occur if one of these commands is used when eight files are already open, or if there is not enough free memory for a buffer to be assigned.

The CATALOG, EXEC, and — commands allocate buffers, use them, and then release them; the OPEN command creates a buffer that exists until it is released by a CLOSE command. Files are not automatically closed when a program comes to an end.

If you get this error, you cannot use any of these commands until you close one of the open files.

This error also occurs if you try to BLOAD a file into the portion of memory used by the system (above HIMEM or below LOMEM).

FILE TYPE MISMATCH (Code 13)

Occurs when a ProDOS command expects to use one type of file, and the specified file is of another type. This message arises from several different incorrect combinations of ProDOS commands with existing file types. Here are the correct combinations.

Command	File Type
CATALOG pn, PREFIX pn	pn must be a directory file (DIR).
LOAD pn, RUN pn, SAVE pn, CHAIN pn	pn must be an Applesoft program file (BAS).
RESTORE pn, STORE pn	pn must be an Applesoft variable file (VAR).
EXEC pn	pn must be a text file (TXT).
OPEN pn, APPEND pn	pn must be a text file (TXT) unless Ttype is used, then file type must match Ttype.
BRUN pn	pn must be a binary file (BIN).
BLOAD pn, BSAVE pn	pn must be a binary file (BIN) unless Ttype is used, then file type must match Ttype.
– pn	pn must be type BAS, BIN, TXT, or SYS.

The file named STARTUP in the volume directory of a startup disk must be of type BAS, BIN, TXT.

PROGRAM TOO LARGE (Code 14)

Occurs when a ProDOS command attempts to place a disk file into the Apple II's memory, and finds the available memory insufficient to contain the entire file. This error can be caused by the CHAIN, LOAD, RESTORE, RUN, or – commands. You (or a previous program) may have set HIMEM too low for the new program to fit.

If you get this error, you can split the program into smaller portions and use the CHAIN command to transfer between one portion of the program and the other.

Remember that a program requires an additional 1K of memory for each file that is simultaneously open.

NOT DIRECT COMMAND (Code 15)

Occurs when you try to use one of the text file commands APPEND, OPEN, POSITION, READ or WRITE from immediate-execution mode. These ProDOS commands can be used only from within PRINT statements in program lines.

SYNTAX ERROR (Code 16)

Occurs when ProDOS encounters a syntax error in a ProDOS command. Check the manual or the help screens for the exact syntax required for the command in question. The problem may be a pathname with illegal characters in it, an incorrect option symbol, a missing option, or a missing or incorrect separator (usually a comma). This message also arises if an option value or command quantity is a negative number or is greater than 65535.

If all ProDOS commands inexplicably cause the `SYNTAX ERROR` message, ProDOS is not started up or is *disconnected* from input and output. To restore, type `CALL 1002` from BASIC (from the Monitor, press CONTROL-C to enter BASIC, then type `CALL 1002`). If this doesn't work, start up the disk again.

DIRECTORY FULL (Code 17)

A ProDOS volume directory file can hold up to 51 files. If a BSAVE, CREATE, OPEN, SAVE, or STORE command indicates a file in a volume directory that already contains 51 files, you get this error. To correct the error, save the file into another directory or onto another disk, then use the Copy File utility to move some files from the volume directory into another directory.

FILE NOT OPEN (Code 18)

Occurs when a command is issued that can only act upon an open file. This error can be caused by the POSITION, READ, and WRITE commands. You must open a file before using any of these commands.

DUPLICATE FILENAME (Code 19)

Occurs when you CREATE or RENAME a file using a pathname that already exists.

FILE BUSY (Code 20)

Occurs when you CAT, CATALOG, DELETE, or RENAME a file that is already open. You must close a file before using one of these commands on that file.

FILE(S) STILL OPEN (Code 21)

Occurs when program execution is interrupted while one or more files are still open (for example, by another error or CONTROL-C). You must close all open files before you issue another LOAD or RUN statement.

For Assembly-Language Programmers: When your assembly-language program generates an error, you can use BASIC.SYSTEM to print the error message. First, have the error code in the accumulator. Next, do a JSR to the PRINTERR routine (BEOC). See the *ProDOS Technical Reference Manual* for more information.

Appendix D

Extras

About This Appendix

This appendix describes the useful programs that are stored in the directory /EXAMPLES/EXTRAS. They perform the following functions:

- TIME allows you to read and set the system date and time so that your files are marked with the proper date.
- READ.DIRECTORY is an example of how to read from a directory file.
- APA, the Applesoft Programmer's Assistant, provides a number of functions that make programming easier and faster.

Using the System Date and Time

ProDOS has two memory locations that contain the current date and time. If you have a clock/calendar card, instructions for making it work with ProDOS are in the section "Using a Clock/Calendar Card" in Chapter 9.

If you don't have a clock/calendar card, ProDOS takes whatever date and time are stored in the system date and time locations, and marks all created and modified files with that time. The program TIME is an Applesoft program that allows you to read and set the system date and time locations so that your files are marked with the current date and time.

Using TIME

With the /EXAMPLES disk in any drive, type the command

```
RUN /EXAMPLES/EXTRAS/TIME
```

and the values of the date and time locations in memory are displayed. If there is no time currently set, the messages `<NO DATE>` and `<NO TIME>` are displayed. The program asks you if you want to update the system date and time. If you say *yes*, you must enter the date in the form

```
DD-MMM-YY
```

where DD = 01 to 31, MMM = JAN to DEC, YY = 00 to 99, and enter the time in the form

```
HH:MM   AM
```

where HH = 01 to 12, MM = 00 to 59, AM = AM or PM. When entering days, hours, or minutes less than 10, you must type in the leading 0.

Note

To set the time on a clock/calendar card, refer to the manual for the card.

If you have a clock/calendar card, this program does not set the time on the card.

Reading From ProDOS Directories

Like all other ProDOS files, directory files can be opened and read. When you read from a directory file, ProDOS automatically interprets the information in that file, and passes it to you in an understandable and familiar form—it gives you the same lines of text displayed by the CATALOG command. For example, to list the /EXAMPLES directory, you can use

```
5 D$=CHR$(4)                          D$ is (CONTROL)-(D).
10  PRINT D$;"OPEN /EXAMPLES,TDIR"    Open directory.
20  PRINT D$;"READ /EXAMPLES"         Prepare to read
30  INPUT L1$: PRINT L1$                directory name,
40  INPUT L2$: PRINT L2$                title line,
50  INPUT L3$: PRINT L3$                blank line,
60  INPUT L4$: PRINT L4$                lines of directory,
70  IF L4$ < > "" THEN GOTO 60          until a blank is read.
80  INPUT L5$: PRINT L5$              Print block use
90  PRINT D$;"CLOSE /EXAMPLES"        and close directory.
```

The first line returned is the name of the directory being read. If it is a volume directory, it is preceded by a slash. The next line read is the title line, shown below. The third line is always empty. Subsequent lines, until the next blank line, are the files in the directory. The block count is the last line read. This program is stored in the file /EXAMPLES/EXTRAS/READ.DIRECTORY.

If you want to do interesting things with the string you have just read (such as write a program that lets you look through a disk's directories), you need to know the exact format of the returned string. A sample line from a directory looks like this:

```
 NAME          TYPE  BLOCKS  MODIFIED          CREATED           ENDFILE SUBTYPE

*FILE.NAME      BAS      14  30-NOV-83  4:33  29-NOV-83 19:11      2001
```

The specific contents of each character of a line read from a directory are listed in Table D-1.

Table D-1. Directory Line Composition

Column	Use
1	Locked or unlocked
2-16	Filename
18-20	File type
23-28	Blocks used by file
31-39	Date file was last modified
41-45	Time of last modification (24 hour clock)
48-56	Date file was created
58-62	Time file was created (24 hour clock)
64-71	Logical end of file
73	Subtype identifier: A = load Address R = Record length
75-79	Load address (hexadecimal)
76-79	Record length (decimal)

The Applesoft Programmer's Assistant (APA)

The Applesoft Programmer's Assistant is a binary program named APA. It is in the EXTRAS subdirectory on the *ProDOS BASIC Programming Examples* disk.

Overview

APA can save a lot of time when you write or change Applesoft programs. The table below lists APA's functions and the commands that perform them. Each command is discussed on the pages that follow.

Function	Command
Automatic Line Numbering	AUTO
Turning Off Automatic Line Numbering	MANUAL
Renumbering a Program	RENUMBER
Putting a Program on Hold	HOLD
Merging Two Programs Into One	MERGE
Deleting Remarks From a Program	COMPRESS
Displaying Control Characters	SHOW
Suppressing Control Characters	NOSHOW
Calculating a Program's Length	LENGTH
Producing a Cross-Reference Listing	XREF
Converting Decimal to Hex and Hex to Decimal	CONVERT
Clearing the APA Program From Memory	EXIT

▲Warning Some APA commands (AUTO, RENUMBER, HOLD, MERGE, COMPRESS, XREF) clear all variables in the program in memory.

Starting APA

With the *ProDOS BASIC Programming Examples* disk in drive 1, and the BASIC prompt on the screen, type

```
-/EXAMPLES/EXTRAS/APA
```

and press RETURN.

The drive whirs for a moment, while ProDOS relocates, loads, and initializes APA.

By the Way: APA is loaded just below HIMEM, and HIMEM is reset to just below APA. Although they are both in memory at the same time, APA and any Applesoft program you load are kept separate and can't interfere with each other. If your program is extremely large, you may need some of the memory taken up by the APA, but this will rarely be a problem.

After APA is loaded, the APA startup display appears:

```
APPLESOFT PROGRAMMER'S ASSISTANT
VERSION n.n
COPYRIGHT APPLE COMPUTER, 1979-83

]_
```

Loading APA doesn't erase any other program you had in memory. So, before starting a *new* program, type **NEW** to clear user memory (this will not erase APA).

With the Applesoft **]** prompt on the screen, you can sue any of APA's commands. APA's commands can be used only in immediate mode—they cannot be part of another program. APA accepts commands in uppercase letters *only*.

Automatic Line Numbering

The AUTO command makes it easier and faster to enter programs. It lets you specify

- the starting line number
- the increment between line numbers.

To specify a starting line number of 100 and an increment of 10, type

```
AUTO 100, 10
```

The space after **AUTO** is optional. The starting line number and increment must be numeric integers in the range 1 to 63999.

and press (RETURN). When you then press (SPACE), the line number **100** appears automatically.

After you type the rest of the Applesoft program line and press (RETURN) and (SPACE), the next line number, **110**, appears.

To specify a starting line number of 100, without specifying an increment, simply type

```
AUTO 100
```

and press (RETURN). The APA sets the increment to 10.

If you wish to leave a line number unused, press (RETURN) then (SPACE) without typing an Applesoft program line—the next line number appears, and no statement is entered under the previous one.

If you want to use a line number that the AUTO function doesn't provide (for example, to use the line number 15 when the starting line number is 10 and the increment is 10), just type **15** instead of pressing (SPACE).

If you change your mind about a statement while you are typing it, press (CONTROL)-(X). When you then press (SPACE) the same number reappears without the statement.

Since a line number appears only if you press (SPACE) after the] prompt, you can type a run-time command any time you see the prompt, as long as you don't begin it with a space. This means you can RUN or LIST your program, or SAVE it and LOAD another, clear the screen with HOME, or use CAT to see the catalog—just as if APA weren't there.

▲Warning It is possible to overwrite APA with run-time commands. When APA is loaded, don't run a program that changes HIMEM or the I/O hooks. And don't press (RESET).

Turning Off Automatic Line Numbering

To turn off automatic line numbering, type

```
MANUAL
```

and press (RETURN). MANUAL is the default when you load APA. To get automatic line numbering, you must use the AUTO command.

Renumbering a Program

The RENUMBER command renumbers the lines in all or part of a program in memory. You specify the starting line number and increment.

Even better, this command also changes line references in GOTO, GOSUB, and ONERR statements! So if the program runs correctly before renumbering, it runs correctly after renumbering—unless you change the order of the lines. RENUMBER does not, however, change line references in REM statements, so check these yourself.

By the Way: If a RENUMBER command would cause interleaving of lines or duplication of line numbers, APA does not execute it. This would happen if a range of lines were renumbered so that its new numbers overlapped the numbers of another range of lines. The resulting error message is **INTERLEAVED OR DUPLICATE LINE NUMBER**.

Let's start with an example. First, type **NEW** to clear memory (this won't affect the APA program). Then enter this simple program at the keyboard:

```
10 PRINT "S"
20 PRINT "A"
30 PRINT "M"
40 PRINT "P"
50 PRINT "L"
60 PRINT "E"
```

Save the program by typing **SAVE SAMPLE** and press (RETURN).

To renumber your SAMPLE program, type

RENUMBER

and press (RETURN). Your whole program is renumbered, starting with line number 100 and incrementing by 10. Type **LIST** to see the result:

```
100 PRINT "S"
110 PRINT "A"
120 PRINT "M"
130 PRINT "P"
140 PRINT "L"
150 PRINT "E"
```

If you don't want to start with line number 100 and increment by 10, type, say

RENUMBER 1000, 50

This results in

```
1000 PRINT "S"
1050 PRINT "A"
1100 PRINT "M"
1150 PRINT "P"
1200 PRINT "L"
1250 PRINT "E"
```

Load the SAMPLE program you saved earlier. To renumber only part of this program, you can specify the first and last old line numbers to be changed. For example, if you type

```
RENUMBER 100, 10, 30, 40
```

where the starting line number is 100, the increment is 10, and the first and last old line numbers are 30 and 40, the result is

```
10 PRINT "S"
20 PRINT "A"
50 PRINT "L"
60 PRINT "E"
100 PRINT "M"
110 PRINT "P"
```

Only the lines previously numbered 30 and 40 have been renumbered. Notice that the lines are in a new sequence, in keeping with their new line numbers. You can use RENUMBER to move subroutines around in a program.

You can specify the first one, the first two, the first three, or all four RENUMBER parameters. RENUMBER uses a default value for omitted parameters. The default values are:

<starting line number>	100
<increment>	10
<first line>	0
<last line>	63999

Thus, these two lines are equivalent:

```
RENUMBER 20, 20
```

```
RENUMBER 20, 20, 0, 63999
```

The RENUMBER command does not let you give two lines the same number or interleave two sets of lines. Instead, you see the message `INTERLEAVED OR DUPLICATE LINE NUMBER`. For example, if you load SAMPLE and type

```
RENUMBER 10, 5, 40
```

you see the error message because this command would not only put line 50 between lines 10 and 20, but also put lines 40 and 60 on top of lines 10 and 20.

Putting a Program on Hold

Use the HOLD command to shelter a program above HIMEM, where it can't be erased by the loading of another program. This lets you load a second program into memory, then use the MERGE command (described in the section "Merging Two Programs Into One") to combine the two programs.

With a program in memory, type

```
HOLD
```

and press (RETURN). You see the message `PROGRAM ON HOLD`.

▲Warning Putting a program on hold does not delete it from the Applesoft work area. You still have to use NEW before starting another program.

Merging Two Programs Into One

After putting one program on hold, you can use the MERGE command to combine it with another.

Let's work with the SAMPLE program you created earlier. First, load SAMPLE and list it on the screen.

```
10 PRINT "S"
20 PRINT "A"
30 PRINT "M"
40 PRINT "P"
50 PRINT "L"
60 PRINT "E"
```

Now delete lines 10, 30, and 50 by typing

```
10
30
50
```

Press (RETURN) after each line number.

When you list the program, you have

```
20 PRINT "A"
40 PRINT "P"
60 PRINT "E"
```

Save this shorter program under the name APE. Put APE on hold by typing

```
HOLD
```

Now reload SAMPLE, and delete lines 20, 40, and 60 from it. This lists as

```
10 PRINT "S"
30 PRINT "M"
50 PRINT "L"
```

Save this program under the name SML. Now type

```
MERGE
```

and press (RETURN); then type

```
LIST
```

and press (RETURN). The result is

```
10 PRINT "S"
20 PRINT "A"
30 PRINT "M"
40 PRINT "P"
50 PRINT "L"
60 PRINT "E"
```

Presto! The program you carved up so diligently is now whole again, and sorted by line number.

If you renumbered SML before merging, for example, by typing

```
RENUMBER 100, 200
```

so that SML looked like this

```
100 PRINT "S"
300 PRINT "M"
500 PRINT "L"
```

you would get a different result. If you typed

```
MERGE
```

you'd get

```
20 PRINT "A"
40 PRINT "P"
60 PRINT "E"
100 PRINT "S"
300 PRINT "M"
500 PRINT "L"
```

The MERGE command, unlike the RENUMBER command, can create duplicate line numbers. For example, put APE on hold, then load SML. Renumber it by typing

```
RENUMBER 20, 10
```

and the listing is:

```
20 PRINT "S"
30 PRINT "M"
40 PRINT "L"
```

Both APE and SML now contain lines numbered 20 and 40.

Now type

```
MERGE
```

and you see the message

```
DUPLICATE LINE NUMBER
20
40

CONTINUE?
```

If you press (Y) to continue in spite of the duplicate line numbers, the line from the HOLD area has priority, and the other line with the same number is deleted. The listing is then

```
20 PRINT "A"
30 PRINT "M"
40 PRINT "P"
60 PRINT "E"
```

If you try to use MERGE when there is no program on hold, you get the message `NO HOLD FILE`.

The MERGE command can perform wonders, and save you a lot of time. But it can also wreak havoc. Before typing the MERGE command, save each of the programs to be merged.

Deleting Remarks From a Program

The COMPRESS command removes documentation remarks (program lines that begin with REM) from the program in memory. COMPRESS lets you maintain two versions of a program. The one that contains REM lines is easier to maintain, and the compressed version runs faster and uses less memory.

To use this feature, load a documented program, then type

```
COMPRESS
```

and press (RETURN). The program in memory is compressed, and can then be saved under a *different name*. A message tells you how many bytes are saved. If you later revise the program, change the documented version and make a new compressed version from it.

Displaying Control Characters

Use the SHOW command to make the control characters in your program visible. Type

```
SHOW
```

Control characters are characters produced when you press (CONTROL) together with some other key.

and press (RETURN). If the program contains any control characters, they are displayed in inverse video when you list the program.

Suppressing Control Characters

After using SHOW to make control characters visible, use the NOSHOW command to make them invisible again. Type

```
NOSHOW
```

and press (RETURN). Control characters are no longer displayed.

Calculating a Program's Length

Use APA's LENGTH command to determine the length of the program in memory. Type

```
LENGTH
```

and press (RETURN). The program's length in bytes is displayed, in both decimal and hexadecimal form.

Producing a Cross-Reference Listing

The XREF command produces an alphabetical cross-reference listing of the Applesoft program in memory. This lists each variable in the program, together with the numbers of all lines in which the variable appears. Load the program you wish to cross-reference, then type

```
XREF
```

and press (RETURN). After a pause that depends on the length of the program, the variables and their line numbers are listed on the screen. Note that all variable names are shortened to two characters (Applesoft distinguishes only the first two characters of a variable name).

Five kinds of variables are identified by a suffix.

$	represents a string
(	represents an array
$(	represents a string array
%	represents an integer variable
%(	represents an integer array

You can interrupt the cross-reference listing by pressing (CONTROL)-(S). To resume an interrupted listing, press (CONTROL)-(S) again.

Converting Decimal to Hex and Hex to Decimal

Use the CONVERT command to convert decimal numbers to hexadecimal and hexadecimal numbers to decimal.

For example, to convert decimal 255 to hexadecimal, type

```
CONVERT 255
```

and press (RETURN). The program immediately responds by displaying

```
255 ($00FF)
```

where **00FF** is the hex equivalent of decimal 255.

To convert hexadecimal 2B to decimal, type

```
CONVERT $2B
```

and press (RETURN). The program responds with

```
43 ($002B)
```

where **43** is the decimal equivalent of hex 2B.

Clearing the APA Program From Memory

To unlink APA commands and return to the system all the memory that was being used by APA, type

```
EXIT
```

and press (RETURN). Once you've used the EXIT command, typing any more APA commands results in a **?SYNTAX ERROR**.

To reload APA, type

```
-/EXAMPLES/EXTRAS/APA
```

and press (RETURN).

Glossary

This glossary defines the terms used in this manual as they apply to ProDOS. Refer to other sources for more complete definitions.

address: A number that specifies a single byte of memory. Addresses can be given as decimal integers or as hexadecimal integers. A 64K system has addresses ranging from 0 to 65535 (in decimal) or from $0000 to $FFFF (in hexadecimal).

APPEND: Attach to the end of. The APPEND command is used to write new data to the end of an existing file.

ASCII: An acronym for the American Standard Code for Information Interchange. This code assigns a unique value from 0 to 127 to each of 128 numbers, letters, special characters, and control characters. It is the code with which the Apple II represents the characters entered at the keyboard.

back up: To make a spare copy of. It is a good habit to back up important files and disks frequently.

binary: Encoded using the base-two numbering system consisting of the two digits, 0 and 1. A single binary digit, a 0 or a 1, is called a **bit**.

binary file: A file whose data are to be interpreted in its binary form. Machine-language programs and pictures are stored in binary files. In comparison, the data in a text file is interpreted as a set of characters. A ProDOS binary file is indicated in a catalog by the abbreviation **BIN**.

BLOAD: Binary load. The BLOAD command causes the binary form of a file to be placed in memory. If the file is not a binary file, its uninterpreted image is placed in memory.

block: 512 bytes of data. This is the unit of storage used by ProDOS. ProDOS regards the information stored on disk and in memory as collections of blocks.

BLOCKS: When you use the CAT or CATALOG command, the column on the screen labeled **BLOCKS** lists the number of blocks of disk space occupied by each file in that directory.

BRUN: Binary run. The BRUN command causes a binary program to be brought into memory and run.

BSAVE: Binary save. The BSAVE command causes the binary data in a portion of memory to be saved in a disk file. If the file is not a binary file, the data is not automatically encoded before being placed in that file.

buffer: A temporary storage area. ProDOS uses a file buffer as a temporary resting place for the characters being read from or written to the file.

BYE: This command allows you to leave Applesoft BASIC, and start another application program, or a program selector, without going through the startup procedure.

byte: A unit of computer memory. A byte is eight bits (Binary digITS) long, and is thus capable of expressing a range of numbers from 0 to 255 (2 to the 8th power is 256). Each character in the ASCII code is represented within a single byte.

CAT or CATALOG: These commands cause a list of the names and characteristics of all the files in a directory to be displayed. This display of information is often referred to as a catalog. CAT displays a 40-column list; CATALOG, an 80-column list.

CHAIN: The CHAIN command runs a BASIC program without first erasing the variables currently in memory.

CHR$: This Applesoft function, when given an ASCII code, returns the character represented by that code. CHR$(4) returns a (CONTROL)-(D).

CLOSE: This command must be issued when you finish using a file. It writes all unwritten data to the file, and it releases the file buffers allocated to that file.

(CONTROL)-(D) This character must precede every ProDOS command used in a program. (CONTROL)-(D) has the ASCII code 4, thus it can be generated using the Applesoft function call, CHR$(4).

(CONTROL)-(RESET:) This combination of keystrokes usually causes an Applesoft program or command to stop immediately. If a program disables the (CONTROL)-(RESET) feature, you need to turn the Apple II off to get the program to stop.

CREATE: This command creates a new file. When used, it places a new file of a designated type into a designated directory.

DASH (—): This command runs a BASIC, machine-language, EXEC, or system program.

DELETE: This command removes a file from its directory. A deleted file cannot be recovered.

directory file (type DIR): A file that contains the names and locations on the disk of other files. Related files should be grouped together into a single directory file. See also volume directory.

disk: A flat circular piece of plastic or metal, coated with a fine metallic powder, onto which information is magnetically recorded.

disk drive: A device that can read information from and record information on a disk. ProDOS lets the Apple II communicate with all disk drives manufactured by Apple Computer, Inc. for the Apple II.

element: As defined in this manual, a string of characters, terminated by a comma or a carriage return, that can be read using the BASIC INPUT statement. For example, INPUT A$,B$ reads two elements.

ENDFILE: End of file. When you display a disk's catalog, the column of information labeled ENDFILE tells the number of bytes that each file would occupy if all the disk space allocated to that file were filled. Refer to Chapter 3 for more details.

/EXAMPLES: The volume name of the disk that contains the ProDOS program and the examples for this manual. This disk contains the version of the system file that must be on every ProDOS startup disk.

EXEC: This command causes input to be taken from a sequential text file rather than from the keyboard. When you use EXEC, you control the operation of the Apple II by using commands that are stored in a text file.

field: In a file, a string of characters preceded by a carriage return character, and terminated by a carriage return character. A field is written to a file by each PRINT statement not terminated by a semicolon. The INPUT command reads an entire field from a file.

file: A file is a named, ordered collection of information on a disk. When you use ProDOS to place information on a disk, you give the file a name and a type. The file's type determines how information is encoded in that file.

filename: The name that identifies a file. A ProDOS filename has a maximum of 15 characters. It can contain letters, digits, and periods, but it must begin with a letter.

firmware: See read-only-memory.

FLUSH: Send unwritten data to its file. Use this command to ensure that the data in a file is identical to the data written to the file. FLUSH is like CLOSE, except the file remains open.

format: To prepare the magnetic surface of a disk for the storage of information. With the *ProDOS User's Disk* or *System Utilities* disk, you can format all types of disks. This utility replaces the DOS command INIT.

FRE: This command is used to access the ProDOS fast housekeeping routines.

HELPSCREENS: A file, stored on the /EXAMPLES disk, that contains all the help screens. Each screen is stored in a single 512-byte record of a random-access text file. For the HELP command to be usable, this file must be on the disk from which ProDOS was started up, and the command **- HELP** must have been previously issued.

hexadecimal: Encoded using the base-16 numbering system. Hexadecimal numbers are formed using the ten digits 0 through 9 and the six uppercase letters A through F. All hexadecimal numbers used with ProDOS must be preceded by the symbol $.

housekeeping: Condensing essential information in memory, thus freeing up as much memory as possible. Sometimes called "garbage collection."

IN#: This command designates the source of subsequent input characters. It can be used to designate a device in a slot or a machine-language routine as the source of input.

input routine: A machine-language routine that performs the reading of characters. The standard input routine reads characters from the keyboard. A different input routine might, for example, read them from an external terminal.

language card: An Apple II interface card that, when placed in slot 0 of a 48K Apple II, gives the Apple II access to a total of 64K of memory. If you have an Apple II or Apple II Plus, you need a language card, or the equivalent, to use ProDOS.

LOAD: This command brings a BASIC program into memory from a file. It clears the current BASIC program and variables from memory and brings in the new program.

load address: The first address in memory from which data was saved into a file with the BSAVE command. When that file is loaded with the BLOAD command or run with the BRUN command, it is placed in memory starting at the load address unless you specify otherwise. The load address of a binary file is listed in the column labeled **SUBTYPE** when you display a catalog of files.

LOCK: This command protects a file from being accidentally renamed, deleted, or altered.

machine-language interface (MLI): The set of machine-language routines, stored in the file named PRODOS, with which ProDOS talks to disk drives. The *ProDOS Technical Reference Manual* contains a full explanation of the ProDOS machine-language interface.

NAME: When a catalog of files is displayed on the screen, the **NAME** column contains the names of the files in the listed directory.

OPEN: This command allocates space in memory for a file's buffers, and it sets the file position pointer to the beginning of the file. The next file-related command to be issued must be a READ or a WRITE. All files opened must be closed.

option: An item in the syntax of a ProDOS command that determines a single aspect of the command's action, such as a pathname or a file type. Unbracketed options must be included each time the command is used, bracketed options can be specified as needed, and two options separated by a vertical line are alternates.

output routine: A machine-language routine that performs the sending of characters. The standard output routine writes characters to the screen. A different output routine might, for example, send them to a printer.

partial pathname: A portion of a pathname. A partial pathname does not begin with a slash, and it does not have to (but may) begin with the name of a volume directory. When you use a partial pathname in a command, the prefix is usually attached to the front to form a full pathname. A partial pathname can be no more than 64 characters long.

pathname: A series of filenames, preceded and separated by slashes, that indicates the entire path, from volume directory to file, that ProDOS must follow to find that file. A pathname used in a command can contain no more than 64 characters, slashes included. (The pathname formed by the prefix and a partial pathname can be up to 128 characters long.)

POSITION: This command causes a specified number of fields to be read and discarded from an open file. It is used to move the position of the file pointer forward in the file.

PR#: This command sends output to a slot or to a machine-language program. It specifies an output routine in the ROM on a peripheral card or in a machine-language routine in RAM by changing the address of the standard output routine used by the Apple II.

prefix: A pathname set to indicate a specific directory file. When you use a partial pathname, the prefix is added to the front of it. You set the value of the prefix using the PREFIX command. A prefix can be no more than 64 characters long, including slashes.

ProDOS command: Any one of the commands recognized by ProDOS. Each has its own syntax, all can be used within programs, and all but five (text file commands) can be used from immediate mode. ProDOS commands used from within programs must be issued as part of a PRINT command string, and must be preceded by CONTROL-D.

/RAM: The volume name of a small volume automatically placed by ProDOS in the alternate 64K of an Apple IIc, or an Apple IIe with an Extended 80-Column Text Card. It can be used just like a disk volume; however, the information stored on it disappears when the computer is turned off.

random-access memory (RAM): This is the readable and writable memory of the Apple II. Its contents are usually filled with programs from a disk, and they are lost when the Apple II is turned off. An Apple II must have 64K of RAM to use ProDOS.

random-access text file: A text file that is partitioned into an unlimited number of uniform-length compartments called records. When you open a random-access text file for the first time you must specify its record length. No record is placed in the file until written to. Each record can be individually read from or written to, hence the name, random-access.

READ: This command, when used after the OPEN command, prepares a file to be read. It can also select the position in the file (record, field, and byte) of the next piece of information to be read. Until the next ProDOS command is issued, subsequent INPUT statements are satisfied by data from the file.

read-only memory (ROM): In the context of this manual, ROM refers to semiconductor chips in the Apple II or on peripheral cards that contain programs essential to the system's operation. The contents of ROM are permanent and unalterable. The Apple II comes with ROM chips that contain the system Monitor and a version of BASIC, and the ROM on a disk controller card contains programs that let the Apple II communicate with one or two disk drives. ROM is sometimes called firmware.

record: A unit of storage in a random-access text file. Every random-access text file can contain a very large number of records; each record holds the exact same number of characters. A program specifies a file's record length (in bytes) when the file is first opened; it must subsequently read and write data into specific records within the file.

record length: The length of a random-access text file's records in bytes. The maximum record length is 65535 bytes; the minimum is 1.

RENAME: This command allows you to change the name of a file. You cannot use this command to move the file from one directory to another, only to change its name *within* a directory. A file must be unlocked to be renamed.

RESTORE: This command clears the BASIC variables currently in memory, and it reads in a new set of variables from a variable file (type VAR). See also STORE.

RUN: This command clears the current BASIC program and variables from memory, brings a BASIC program into memory from a file, and runs it. An option lets you specify the first program line to be run.

SAVE: This command lets you save the BASIC program currently in memory as a BASIC program file (type BAS).

sequential-access text file: A text file made up of a sequence of fields. A field is a string of characters terminated by a carriage return character. Sequential text files are best used for types of data that will be stored and retrieved sequentially.

slots: The narrow sockets inside the Apple II, II Plus, and IIe computers, where you can install peripheral cards. The Apple IIc has no slots, but is the equivalent of a IIe with five peripheral cards built in.

start up: To get the system running. In the context of ProDOS, starting up is the process of reading the ProDOS program (in the files PRODOS and BASIC.SYSTEM) from the disk, and running it.

startup disk: A disk that contains all the information needed to get the computer running. A ProDOS startup disk must be formatted using the ProDOS Filer, and it must contain the files PRODOS and BASIC.SYSTEM.

STORE: This command causes the BASIC variables currently in memory to be arranged in a compact form and then placed in a BASIC variable file (type VAR). The variables so stored can be returned to memory using the RESTORE command.

SUBTYPE: In a catalog, the column labeled **SUBTYPE** contains two types of information: for a random-access text file, the file's record length (R) in decimal; and for a binary file, the file's load address (L) in hexadecimal.

syntax: A representation of a command that specifies all the possible forms the command can take. The syntax of each ProDOS command is given as a command word followed by a list of options.

SYSTEM: A file with a name of the form XXX.SYSTEM must be in the volume directory of every startup disk; it contains the system program that is run when the disk is started up. On the /EXAMPLES disk, BASIC.SYSTEM contains the ProDOS BASIC program; on the disk named /USERS.DISK, FILER.SYSTEM contains the ProDOS Filer

text file (type TXT): A file whose contents are interpreted as characters encoded using the ASCII format. ProDOS defines two types of text files: sequential, a grouping of sequentially accessible fields of text; and random-access, a collection of equal-sized, and independently accessible, groups of characters.

TYPE: In a catalog, the column with this heading names the type of each file listed. Types are given as three-letter abbreviations. There is a list of file type abbreviations in Table A-2.

UNLOCK: This command reverses the effect of the LOCK command. A file must be unlocked if it is to be renamed, deleted, or altered.

VERIFY: Before you create a file with a particular name, this command allows you to check whether there is already a file with that name.

volume: A source or destination of information. As used in this manual, volume always refers to a disk. It could also, for example, refer to a magnetic tape or a location in a network.

volume directory: The main directory of a volume. On a disk, the volume directory is a file that contains the names and locations on that disk of up to 51 other files, any of which may themselves be directory files.

WRITE: This command, when used after the OPEN command, prepares a file to be written to. Until the next ProDOS command is issued, all subsequent PRINT statements send characters to this file.

write-protected: A disk drive that uses flexible disks can only write on a disk that has a small notch in the proper location. If this notch is covered, or if the notch does not exist, the disk is write-protected. The notch itself is referred to as a write-enable notch.

Index

A

address(es)
  binary 160
  load 187
  memory 160
  option 184
ADDRESS
  file 130
address option 184
allocation buffers 202
APA (Applesoft Programmer's Assistant) program 234, 237
APPEND command 73, 93, 120-121, 140-142, 149, 186, 201
  DOS-ProDOS differences 213
Apple II Plus xxviii, 2
Apple IIc xxvii, xxviii, xxix, 2
  ports xxix
Apple IIe xxviii, 2
Apple III SOS 178
Applesoft xxvii
  BASIC xxviii
    leaving 197
    ProDOS changes 217-219
    Programmer's Assistant 234, 237
array 78
assembly language programs 175
AUTO command 238
automatic line numbering 238
AWAY program 149-150

B

backing up disks 3
BASIC xxvii
  Applesoft xxvii, xxviii
  Integer xxviii, 212
  leaving 89, 197
  listing as text files 151
  ProDOS changes to Applesoft 217-219
  program files 52
  prompt 6
  tokens 166
  variables 194-196
BASIC.SYSTEM 175
binary
  addresses 160
  commands 159-169, 204-206
  data 162
    loading 157, 164
    saving 157, 167
  files 156, 159, 204-206
    naming 162
  pictures 159
  programs 156
    running 157, 162
    saving 167
BLACK.BOOK program 127-134
BLOAD command 157, 159, 206
  DOS-ProDOS differences 213
blocks, free 187
BLOCKS FREE message 39
BLOCKS USED message 39
boot *See* startup
bracketed options 22
brackets 22
BRUN command 53, 157, 159, 162, 191, 204
  DOS-ProDOS differences 213
BSAVE command 157, 159, 167, 204-205
  DOS-ProDOS differences 214
buffer(s)
  allocation 202
  file 115, 135, 137
built-in drive xxix
BYE command 67, 89, 197, 217
bytes, number of options 183

C

CAPTURE program 151, 152
card(s)
  clock/calendar 38, 174, 234
  80-column 85
  Extended 80-Column Text 16
  peripheral xxix, 5
carriage return 98, 125, 126
  character 100
CAT command 186, 216
CATALOG command 29, 138, 186
  DOS-ProDOS differences 214
CHAIN command 65, 77, 78-80, 183, 231
  DOS-ProDOS differences 214
chaining 211
character(s)
  reading from text file 104
  storing in fields 100
  writing to text file 103
clearing BASIC variables 195
clock/calendar card 38, 174, 234

CLOSE command 92, 101, 117, 137, 166, 198-199
closing the notebook with 96, 98
creating an EXEC file using BASIC 149, 153
DOS-ProDOS differences 214
rolling the scroll with 94, 95
combining programs with EXEC 152
command(s)
APPEND 73, 93, 120-121, 140-142, 149, 186, 201, 213
binary 159-169, 204-206
BLOAD 157, 159, 206, 213
BRUN 53, 157, 159, 162, 191, 204, 213
BSAVE 157, 159, 167, 204-205, 214
BYE 67, 89, 197, 217
CAT 186, 216
CATALOG 29, 138, 186, 214
CHAIN 65, 77, 78-80, 183, 214, 231
CLOSE 92, 94, 95, 96, 98, 101, 117, 137, 149, 153, 166, 198-199, 214
commands not in DOS 215-217
COMPRESS 245
CONT 166
CONVERT 246
CREATE 29, 41-43, 188, 216
DASH 53, 54-55, 159, 162, 191, 217
DELETE 29, 45-46, 189
DOS 211, 213-215
EXEC 53, 102, 146-153, 191, 203, 227
FLUSH 93, 121-122, 142, 201, 201-202, 216
FRE 67, 77, 88, 197, 217, 219
HELP 73
HGR 166, 218
HGR2 166, 218
HOLD 242
IN# 67 77, 87-88, 158, 196, 214, 219
INIT 208
INPUT 95, 98, 218
LENGTH 246
LIST 149
LOAD 53, 58-59, 192
LOCK 29, 46, 60, 189
MERGE 242
Monitor 153
NEW 153
NOSHOW 245
NOTRACE 72, 219
ONERR GOTO 222
OPEN 73, 92, 94, 95, 96, 98, 100, 115-117, 135-136, 149, 197-198, 214
options 180-185
POSITION 73, 93, 143, 202, 215, 227
PR# 67, 77, 85-87, 158, 195, 214, 219
PREFIX 21, 29, 187, 216
deferred mode 73-74
PRINT 94, 96, 103, 140, 149
ProDOS 22
Quit 175
READ 73, 93, 119, 126, 139, 139, 186, 199, 215, 227
RENAME 29, 44-45, 75, 76, 188
RESTORE 66, 77, 81-84, 195, 216
RUN 53, 54-59, 191, 215
SAVE 53, 60-62, 193
SHOW 245
STORE 58, 66, 77, 80-81, 184, 216
structure 210
syntax 10
TEXT 166, 218
TRACE 72, 219
UNLOCK 29, 48, 60, 190
VERIFY 29, 49, 190, 217
WRITE 73, 93, 94, 95, 96, 98, 100, 118, 126, 137, 149, 186, 200, 201, 215
XREF 246
COMPRESS command 245
CONJUGATE program 107, 109
CONJUGEAT program 108
CONJUGEATEN program 107
CONT command 166
control characters 72
displaying 245
suppressing 245
CONTROL-C 153
CONTROL-D 71, 85, 88, 186
controller, disk drive xxix
disk drive xxix
conversion program
DOS-to-ProDOS 209
CONVERT command 246
COUT1 (standard output routine) 172
CREATE command 16, 29, 41-43, 188, 216
cross-reference listing 246
CSWH (Apple II memory locations) 172
CSWL (position in the file pointer) 172
current position 99

D

DASH command 53, 54-55, 159, 162, 191, 217
data
  appending 140-142, 201
  binary 157, 162
    loading 164
    saving 167
  files xxvii
  flushing 142
  integrity 121
  protection 202
date, system 234
DATE locations 174
DATETIME routine 174
debugging 72
decimal-hexadecimal conversion 246
deferred mode 10, 73, 185
DELETE command 29, 45-46, 189
deleting files 45-46
deleting remarks 245
device, peripheral xxix
DIR (directory file) 17
directories 235
  creating 41
  files 17, 235
  volume 15, 16, 178, 179
DIRECTORY FULL message 223, 231
DISK FULL message 223, 229
Disk II drives 11, 208
disk(s) 208, 228
  backing up 3
  DOS 208
  drive xxvii, xxviii, 11
    built-in xxix
    controller xxix
    drive door 228
    Examples xxvii, 2, 3
    formatting 14
    identifying 208
    operating systems xxvii
    ProDOS formatted 208
    ProDOS User's 11, 28
    startup 6, 68-70
    System Utilities xxvii, xxviii, 2, 11, 28, 209
display, 80-column xxix
DOIT.EXEC file 149-150
DOS 11, 72, 208, xxviii
  commands improved in ProDOS 213-215
  commands not supported by ProDOS 211
  differences from ProDOS 210
  disks 208
DOS 3.2.1 208
DOS 3.3 208
DOS-ProDOS Conversion Program 209
drive(s)
  disk xxvii, xxviii, 11, 208
    built-in xxix
    number 24, 182
  startup 68-70
DUPLICATE FILENAME message 223, 232

E

E.S.P. program 82
80-column
  card 85
  display xxix
  loading graphics with 166
elements 103, 108
  multiple per field 107
end address option 184
END OF DATA error message 122, 223, 226
end of file, logical 187
ENDFILE heading 138
entering text 110-114
error
  handling 74-77
  numbers 74
error messages 25
  END OF DATA 122
  FILE BUSY 115
  FILE LOCKED 75, 76193
  FILE NOT OPEN 115
  handling in Applesoft BASIC programs 222
ESPVARIABLES file 83
Examples disk xxvii, 2, 3
EXEC
  command 53, 102, 146-153, 191, 203, 227
  files 146-150
  programs 146
EXEC.DEMO program 147-148
EXIT (from APA) 247
expansion slots xxix
Extended 80-Column Text Card 16
EXTRAS 234

F

field(s) 98, 124
  number of 182
  skipping 143
FILE BUSY message 115, 223, 232

FILE LOCKED message 75, 76 193, 223, 229
FILE NOT OPEN message 115, 223, 232
FILE(S) STILL OPEN message 223, 232
FILE TYPE MISMATCH message 223, 230
file(s)
  ADDRESS 130
  appending to 120-121, 140-142
  as input 203
  binary 156, 159, 204-206
    naming 162
  buffer 115, 135, 137
  closing 117, 137, 198-199
  converting 209
  creating 188
  data xxvii
  deleting 45-46, 189
  directory 17, 41
  duplicating 60
  EXEC 146-150
  flushing 201-202
  input from 146
  locking 46, 189
  name 14, 16, 44-45, 135
  opening 115-117, 135-136, 197, 198
  organization 210
  picture 162
  protecting 183
  protection 45, 46-48, 60
  random-access 95-99, 124-134
  reading 119, 139
  renaming 44-45, 188
  sequential 124
  sequential-access 93-95, 98-99
  size 178
  structures 178
  text 14, 92, 211
    input from 153
    sequential 100-110
  type 14, 43, 185
    option 184
    types 178, 210
      and matching ProDOS commands 230
    unlocking 48, 190
    verifying 49, 190
    writing to 118, 137
filenames 210
  duplicate 231
FLUSH command 93, 121-122, 142, 201, 201-202, 216
formatting disks 14
FOUR.FRUITS text file 105-106
FP command 212
FRE command 67, 77, 88, 197, 217
  ProDOS changes 219
free blocks 187

G

GET statement 104, 108, 109, 139, 227
GET.FRUIT program 106
GET.TEXT program 110, 113-114, 148
GOSUB command 109
graphics
  high-resolution 160
  loading 164
  pages 166
  saving 167
  screens 160
greeting program 212

H

HELP command 73
HELPSCREENS file 11
hexadecimal notation 23
hexadecimal-decimal conversion 246
HGR command 166
  ProDOS changes 218
HGR2 command 166
  ProDOS changes 218
high-resolution graphics pages 166
HIMEM 167
  ProDOS changes 218
  putting program above 242
HOLD command 242
housekeeping 197
  routines 77, 88

I

I/O ERROR message 217, 223, 228
identifier 115
immediate mode 10, 73, 231
improved DOS commands 213-215
IN# command 67, 77, 87-88, 158, 196
  DOS-ProDOS differences 214
  ProDOS changes 219
INIT command 208
input
  from files 146, 153, 203
  getting from slots 196
  reading from slot 158, 170-173
  routine 172
  selecting device 87
INPUT command 95, 98
  ProDOS changes 218
INPUT statement 104, 106, 139, 227
INT command 212

Integer BASIC xxviii, 212
integrity of data 121
interface
machine-language 178
serial xxix
INTERLEAVED OR DUPLICATE LINE NUMBER message 240, 241
interrupt protocol 178
INVALID PARAMETER message 223, 229
I/O ERROR message 217, 223, 228

J, K

KEYIN 172

L

length
of record 125, 136
option 135, 184
LENGTH command 246
line number option 184
line numbering, automatic 238
LIST command 149
LISTFILE 101
LISTING.EXEC file 151
LISTSELF program 101
load address 187
LOAD command 53, 58-59, 192
loading
binary data 164
graphics 164
programs 58-59
location(s)
DATE 174
TIME 174
LOCK command 16, 29, 46, 60, 189
locking files 46
logical end of file 187

M

machine language xxviii, 86, 88
machine-language
interface 178
routines 167
MAKE DIRECTORY 16
MAKE.DOIT program 149-150
MAKE.FRUIT program 105-106
MAKE.TEXT program 110-113, 120, 150
MANUAL command 239
MAXFILES command 211, 212
memory 5, 78, 210
addresses 160
clearing 197
MERGE command 242
mode
deferred 10, 73, 185
immediate 10, 73, 231
Monitor
I/O links 172
input link 172
leaving the 173
output link 172
program 172-173
Monitor commands 153
mouse xxix

N

naming binary files 162
NEW command 153
new ProDOS commands 215-217
NO BUFFERS AVAILABLE message 223, 229
NO DEVICE CONNECTED message 223, 226
NOHELP 11
NO HOLD FILE message 244
NOSHOW command 245
NOT DIRECT COMMAND message 223, 231
notation
hexadecimal 23
NOTRACE command 72
ProDOS changes 219

O

ONERR GOTO statement 74, 76, 114, 222
ONERR.DEMO program 75
OPEN command 73, 92, 94, 95, 96, 98, 100, 115-117, 135-136, 149, 197-198
DOS-ProDOS differences 214
opening files 115-117
operating systems xxvii, 208
option(s) 22-23
address 184
drive number 24, 182
end address 184
file type 184
length 184
line number 184
number of bytes 183
number of fields 182
pathname 23
ProDOS commands 22, 180-185
record number 182
slot number 24, 181, 184

output
directing to devices 85
directing to slot 158, 170-173
routine 172
sending to slots 195

P

partial pathname(S) 20-21, 179
Pascal 208
path 179
PATH NOT FOUND message 223, 228
pathname(s) 21, 181
option 23
partial 20-21, 179
peripheral
card xxix, 5
device xxix
.PIC suffix 162
picture file 162
pictures
binary 159
pointer, position-in-the-file 99
POKER.EXEC program 152
ports xxix
position, current 99
POSITION command 73, 93, 143, 202, 227
DOS-ProDOS differences 215
PR# command 67, 77, 85-87, 158, 195
DOS-ProDOS differences 214
ProDOS changes 219
prefix 20-21, 29, 115, 179, 211
reading 187
setting 187
PREFIX command 21, 187, 216
deferred mode 73-74
PRINT command 94, 96, 103, 118, 137, 140, 149
ProDOS xxvii, 175
command options 180-185
commands and matching file types 230
commands not in DOS 215-217
differences from DOS 210
directories 235
disks 208
pathname 179
syntax 180
using from within a program 71
ProDOS BASIC Programming Examples disk xxvii, 2, 3
ProDOS commands
in programs 71-77
options 22
syntax 22
ProDOS User's Disk xxviii, 11, 28
PROGRAM TOO LARGE message 223, 231
program(s)
ADDRESS 141
APA 234, 237
AWAY 149-150
binary 156
running 157, 162
saving 167
BLACK.BOOK 127-134
CAPTURE 151, 152
chaining 183
combining (with EXEC) 152
CONJUGATE 107, 109
CONJUGEAT 108
CONJUGEATEN 107
EXEC 146
EXEC.DEMO 147-148
files 52
GET.FRUIT 106
GET.TEXT 110, 113-114, 148
greeting 212
length 246
LISTSELF 101
loading 58-59, 192
machine-language 167
MAKE.DOIT 149-150
MAKE.FRUIT 105-106
MAKE.TEXT 110-113, 120, 150
merging 242
Monitor 172-173
POKER.EXEC 152
READ.DIRECTORY 234
running 54-59, 191
saving 193
saving on disk 60
selector 89, 197
system 191
TIME 38, 234
prompt, BASIC 6
protecting files 45, 46-48, 60, 183
protocol
interrupt 178

Q

Quit command 175

R

R (Record length) 39
RAM 16, 17, 62
random-access text files 92, 95-99, 124-134
RANGE ERROR message 223, 225
READ command 73, 93, 119, 126, 139, 139, 186, 199, 227
DOS-ProDOS differences 215

READ.DIRECTORY program 234
reading from a file 104
reading text 110-114
record(s) 98, 124
  length 125, 135, 136
  number 126
  number option 182
  size 126, 136
REM statements, deleting 245
remarks, deleting 245
RENAME command 29, 44-45, 75, 76, 188
RENUMBER command 239
RESTORE command 66, 77, 81-84, 195, 216
restoring variables 81-84
routine(s)
  DATETIME 174
  housekeeping 77, 88
  machine-language 167
  standard input 172
  standard output 172
RUN command 53, 54-59, 191
  DOS-ProDOS differences 215
running binary programs 162
running programs 54-59
running (system) programs 191

S

SAVE command 53, 60-62, 193
saving
  BASIC variables 194-196
  binary data 167
  binary programs 167
  graphics 167
  machine-language programs 167
  programs xxvii
  variables 58, 80-84
sequential-access text files 92, 93-95
sequential text files 100-110, 124
serial interface xxix
setting the prefix 187
SHOW command 245
SHOWOFF.EXEC file 147-148
skipping fields 143
slot(s) xxix, 5, 228
  directing output to 158, 170-173
  getting input from 196
  number option 24, 181, 184
  reading input from 158, 170-173
  sending output to 195
SOS 14, 178, 208
standard
  input routine 172
  output routine 172
startup
  disk(s) 6, 68-70
  drive 68-70
  greeting program 212
  procedure 89
statement(s)
  GET 104, 108, 109, 139, 227
  INPUT 104, 106, 139, 227
  ONERR GOTO 74, 76, 114
  PRINT 118, 137
  REM 245
  STOP 166
STORE command 58, 66, 77, 80-81, 184, 216
string variables 246
subroutines 109
SUBTYPE 39
syntax
  command 10
  ProDOS 22, 180
SYNTAX ERROR message 223, 231
sys file type 175
system
  date 234
  time 234
system programs, running 191
System Utilities disk xxvii, xxviii, 209

T

text
  80-column, loading graphics with 166
  entering 110-114
  files 211
    input from 153
    random-access 124-134
    sequential 100-110, 124
  reading 110-114
  window 112
Text Card
  Extended 80-Column 16
TEXT command 166
  ProDOS changes 218
text file(s) 14, 92
  choosing 98
  input from 153
  sequential 100-110
time
  system 234
TIME
  locations 174
  program 38, 234
tokens
  BASIC 166
TOTAL BLOCKS 39
TRACE command 72
  ProDOS changes 219
transfer rate 178

U

UNLOCK command 29, 48, 60, 190
unlocking files 48
utilities disks xxviii, 2

V

VAR filetype 81
variables 77, 78
  clearing 195
  restoring 81-84
  saving 58, 80-84, 194-196
  types 246
VERB.LIST text file 107, 108, 109
VERIFY command 29, 49, 190, 217
volume
  directory 15, 16, 178, 179, 211
  name 16
  numbers 210-211
  size 178

W

window, text 112
WRITE command 73, 93, 94, 95, 96, 98, 100, 118, 126, 137, 149, 186, 200, 201
  DOS-ProDOS differences 215
WRITE PROTECTED message 223, 226
write-enable notch 226
writing to a file 103

X, Y, Z

XREF command 246
XXX.SYSTEM 175

Notes

Tell Apple About Your .....

☐ Please contact your authorized Apple dealer when you hav
trained by Apple Computer and are given the resources to
you need the name of an authorized Apple dealer in your

☐ Would you like to tell Apple what you think about this pro
product, we would like to hear from you. You can help us
questionnaire below and marking the appropriate boxes on
have more than one response to a question, mark all the b
Apple. Include additional pages of comments if you wish.

1. How would you rate *BASIC Programming with ProDOS 8*?
2. Where did you purchase the manual? (**1** = bookstore, **2** = Ap
3. Did you order the *ProDOS 8 Update* from APDA? (**1** = no, **2**
4. How much experience have you had with computers? (**1** = n
5. Which Apple computer(s) are you using? (**1** = Apple II, **2** = A
5 = Apple IIGS, **6** = Macintosh family)
6. If you program, what other programming languages do you
4 = other)
7. Which of the following best describes your programming?
3 = a hobby, **4** = a source of income, **5** = other)
8. Are you a Certified or Registered Apple developer? (**1** = Certi
9. How useful was the Programming Examples disk? (**1** = not u
10. How much of *BASIC Programming with ProDOS 8* have yo
3 = reference only)
11. How easy was the manual to read and understand? (**1** = diffi
12. How helpful were the illustrations in understanding the mat
(**1** = not helpful . . . **6** = very helpful)
13. How easy was it to find the information you needed? (**1** = di
14. To what degree did the technical material in the manual me
15. What did you like best about the manual?
16. What did you like least about the manual?
17. Please describe any errors or inconsistencies you may have
18. What suggestions do you have for improving *BASIC Progra*

Thanks for your time and effort.

A Pub

Return Address _____

Apple Computer, Inc.
P.O. Box 1143
Cupertino, CA 95014
USA

Place
First Class
Postage
Here



8 3-25-86

BOSTON PUBLIC LIBRARY
3 9999 00602 704 7

WITHDRAWN
This item is the property of the Boston Public Library.
Use of this material benefited the Library.

BASIC PROGRAMMING WITH PRODOS
QA76.8 A662P7/1987X

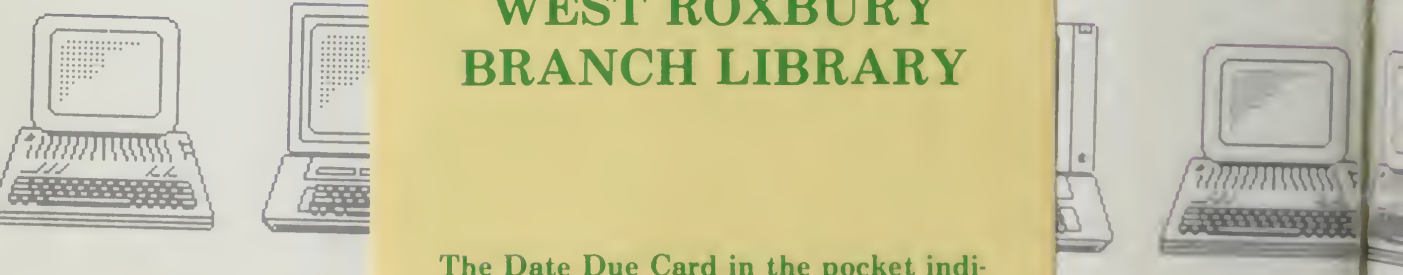
88607798-56

WRx

Boston Public Library

WEST ROXBURY
BRANCH LIBRARY

The Date Due Card in the pocket indicates the date on or before which this book should be returned to the Library.
Please do not remove cards from this pocket.



PLEASE
RETURN

Cut Here



