Apple *II*

DOS Programmer's Manual

For II, II+, //e

Customer Satisfaction

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

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

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

Limitation on Warranties and Liability

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

Copyright

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

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

Product Revisions

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

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

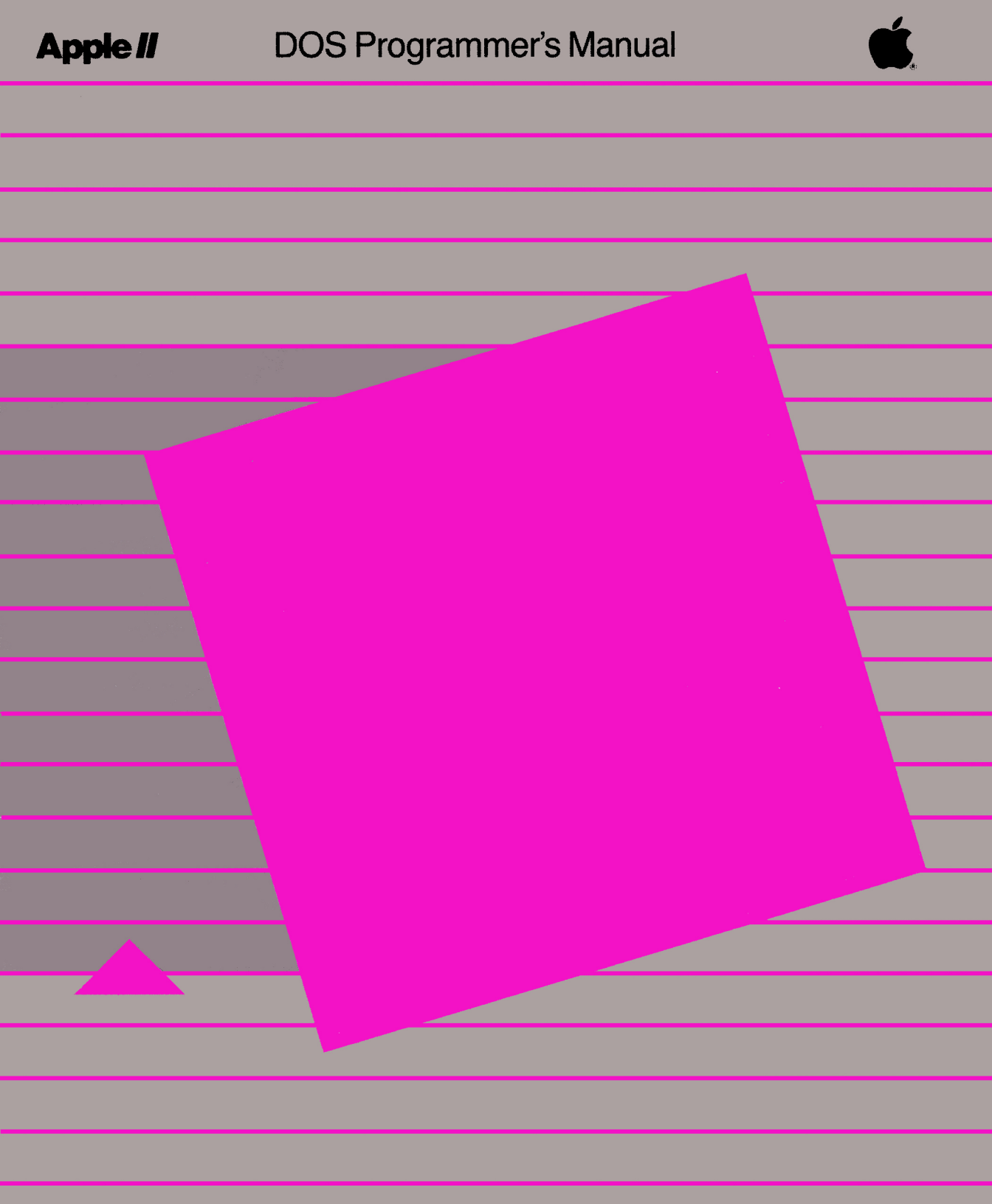
Apple and the Apple logo are registered trademarks of Apple Computer, Inc.

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

Reorder Apple Product #A2L2012

Apple II

DOS Programmer's Manual



Contents

Preface xi

1 Introduction 1

3 1.1 The DOS Environment
3 1.1.1 The DOS Disks
4 1.1.2 Requirements for Using DOS
4 1.1.3 A Few Conventions
5 1.2 How DOS Commands Are Described
5 1.2.1 Notation
5 1.2.2 Syntax
7 1.3 Direct DOS Commands
7 1.3.1 The CATALOG Command
8 1.3.2 The INIT Command
9 1.3.3 The RENAME Command
10 1.3.4 The LOCK Command
11 1.3.5 The UNLOCK Command
11 1.3.6 The DELETE Command
12 1.3.7 The VERIFY Command
13 1.3.8 The RUN Command
14 1.3.9 The LOAD Command
15 1.3.10 The SAVE Command

2 DOS Programming Tools 17

19 2.1 Ways of Using DOS Commands
19 2.1.1 Immediate Execution
20 2.1.2 Deferred Execution
20 2.1.3 Command Files
20 2.2 BASIC Program Files
22 2.2.1 Switching Languages with the FP and INT Commands
24 2.2.2 Returning to BASIC from the Monitor Program

25 2.3 Using DOS From a Program
26 2.3.1 Example
27 2.3.2 Things to Watch Out For
28 2.4 Debugging With the MON and NOMON Commands
29 2.4.1 The MON Command
30 2.4.2 The NOMON Command
30 2.5 Intercepting Errors With Applesoft
31 2.5.1 Example
33 2.5.2 Finding Hidden Characters in a File Name
34 2.6 Talking to Other Devices From a Program
34 2.6.1 Starting a Program With PR# and IN#
35 2.6.2 The PR# Command
36 2.6.3 The IN# Command
36 2.7 The RENUMBER Program
37 2.7.1 RENUMBER Commands
41 2.7.2 RENUMBER Error Messages
43 2.8 Summary
43 2.8.1 Commands
43 2.8.2 Program

3 ***Using Sequential Text Files*** **45**

47 3.1 Comparing Sequential- and Random-Access Text Files
49 3.1.1 Choosing Sequential- or Random-Access Text Files
50 3.1.2 Current Position Pointer
50 3.2 Sequential-Access Text Files
50 3.2.1 The Field
53 3.2.2 Entering and Reading Text
61 3.2.3 Programs: Entering and Reading Text
65 3.3 Commands Used With Sequential-Access Text Files
65 3.3.1 The OPEN Command - Deferred Execution
66 3.3.2 The CLOSE Command - Immediate or Deferred Execution
67 3.3.3 The WRITE Command - Deferred Execution
68 3.3.4 The READ Command - Deferred Execution
68 3.3.5 The POSITION Command - Deferred Execution
69 3.3.6 The APPEND Command - Deferred Execution
71 3.4 Command Summary

4 Using Random-Access Text Files 73

75 4.1 Random-Access Text Files
76 4.1.1 Record Length
76 4.1.2 Writing to a Record
77 4.1.3 Reading From a Record
77 4.2 A Demonstration: The RANDOM Program
79 4.3 A Sample Random-Access Program
79 4.3.1 Controlling the Program
79 4.3.2 Storing Records
80 4.3.3 Writing a Record
82 4.3.4 Reading a Record
84 4.4 Commands Used With Random-Access Text
84 4.4.1 The OPEN Command - Deferred Execution
85 4.4.2 The CLOSE Command - Immediate or Deferred Execution
86 4.4.3 The WRITE Command - Deferred Execution
87 4.4.4 The READ Command - Deferred Execution
88 4.5 Command Summary

5 Programming With More Sophistication 89

91 5.1 Using a Command File
92 5.5.1 The EXEC Demonstration
94 5.1.2 Creating a Command File
96 5.1.3 Special-Purpose Command Files
99 5.2 The EXEC Command
101 5.3 The MAXFILES Command
101 5.3.1 Buffering Information
102 5.3.2 Example
103 5.4 CHAIN
103 5.4.1 The CHAIN Command - Integer BASIC
105 5.4.2 The CHAIN Program - Applesoft
106 5.5 The MASTER Program
107 5.5.1 Example
108 5.6 Making a Turnkey Disk
108 5.6.1 Creating a Turnkey Disk
109 5.6.2 Creating a Master Turnkey Disk
109 5.6.3 A Disk for All Systems
113 5.7 Summary
113 5.7.1 Commands
113 5.7.2 Program

6 Using Memory More Efficiently 115

117 6.1 Binary Files
118 6.1.1 Binary Addresses
118 6.1.2 An and Ln: The Memory Address Arguments
119 6.2 The Binary Commands
120 6.2.1 The BRUN Command
121 6.2.2 The BLOAD Command
122 6.2.3 The BSAVE Command
124 6.3 DOS and the Monitor Program
125 6.3.1 Input and Output Registers of the Monitor Program
126 6.3.2 Input and Output Registers of DOS
127 6.4 Memory Usage and Entry Points
127 6.4.1 PR# and IN# in Memory
129 6.4.2 Memory Maps
133 6.4.3 DOS Entry Points
135 6.5 Binary Command Summary

A Dealing With 13-Sector Disks 139

140 A.1 Converting 13-Sector Disks: The CONVERT13 Program
140 A.1.1 Example
143 A.1.2 The Wildcard Character
144 A.2 Running Unconverted 13-Sector Disks
144 A.2.1 Using the START13 Program
145 A.2.2 Using the BASICS Disk

B The Storage Process 147

147 B.1 Tracks and Sectors
148 B.2 Contents of File Sectors
149 B.2.1 Format of File Sectors According to File Type
149 B.2.2 The Track/Sector List
151 B.2.3 The Disk Directory
155 B.2.4 The Volume Table of Contents
157 B.2.5 The Track Bit Map
158 B.3 Track and Sector Allocation
159 B.4 Addressing Tracks and Sectors: The RWTS Subroutine
160 B.4.1 Example
162 B.4.2 Formats: I/O Block and Device-Characteristics Table
163 B.5 DOS Vectors in Page 3
165 B.6 Zero Page Use

C DOS Error Messages 167

168 C.1 DOS Error Messages
169 C.2 Recovering From Errors

D

Programs 177

177 D.1 Programs on the SYSTEM MASTER Disk
178 D.2 Programs on the SAMPLE PROGRAMS Disk

E

Summary of DOS Operating Concepts and Commands 181

181 E.1 Operating Concepts
181 E.1.1 Cold Start
182 E.1.2 Warm Start
182 E.1.3 Initializing a Disk
182 E.1.4 Creating a Master Disk
183 E.1.5 Interpreting Commands
183 E.1.6 Capacity
183 E.1.7 File Types
183 E.2 Command Notation
183 E.2.1 Conventions
184 E.2.2 Syntax
184 E.2.3 Arguments
186 E.3 Command Summary
187 E.3.1 Nonprogramming Commands for Accessing Disks
190 E.3.2 Commands that Control the Programming Environment
192 E.3.3 Programming Commands for Sequential-Access Text Files
195 E.3.4 Programming Commands for Random-Access Text Files
197 E.3.5 Commands for Binary Files

Glossary 199

Index 211

Figures and Tables

Chapter 1 Introduction

5 Figure 1-1. The Syntax of DOS Commands

Chapter 2 DOS Programming Tools

22 Figure 2-1. FP and INT
37 Figure 2-2. The RENUMBER Title Screen

Chapter 3 Using Sequential-Access Text Files

48 Figure 3-1. Printing to a Sequential-Access Text File
48 Figure 3-2. Printing to a Random-Access Text File

53 Table 3-1. Printing to a Text File
54 Table 3-2. Reading From a Text File

Chapter 4 Using Random-Access Text Files

80 Figure 4-1. Five Addresses in BLACK.BOOK
80 Figure 4-2. Writing an Address to Record 5
82 Figure 4-3. Reading an Address from Record 5

Chapter 5 Programming With More Sophistication

92 Figure 5-1. The First Screen of the EXEC Demonstration
93 Figure 5-2. EXEC DEMO's Final Screen
94 Figure 5-3. A Command File

Chapter 6 Using Memory and Disks More Efficiently

119 Figure 6-1. BRUN, BLOAD, and BSAVE
130 Figure 6-2. Memory Areas Overwritten When Booting DOS
131 Figure 6-3. Memory Areas Used by DOS and Both BASICs

125 Table 6-1. Monitor Input Register in Locations 56-57 ($38-$39)
125 Table 6-2. Monitor Output Register in Locations 54-55 ($36-$37)
126 Table 6-3. DOS Input Register
126 Table 6-4. DOS Output Register
132 Table 6-5. The Values of HIMEM

Appendix A Dealing With 13-Sector Disks

139 Figure A-1. 13-Sector Versus 16-Sector Disks
140 Figure A-2. The CONVERT13 Menu
145 Figure A-3. The START13 Title Screen

Appendix B The Storage Process

148 Figure B-1. Tracks on a Disk
159 Figure B-2. Tracks and Sectors

149 Table B-1. Format of File Sectors for Different File Types
150 Table B-2. First Sector of a Track/Sector List
152 Table B-3. One Sector of a Disk Directory
153 Table B-4. Directory Entry for One File
154 Table B-5. Byte Indicating the File Type
155 Table B-6. Locked/Unlocked Values of the File Type Byte
155 Table B-7. Volume Table of Contents: VTOC (Track $11, Sector $Ø)
157 Table B-8. Track Bit Map for One Disk Track
157 Table B-9. Typical Track Bit Map
162 Table B-10. Format of an I/O Block
163 Table B-11. Format of a Device-Characteristics Table
164 Table B-12. DOS Vectors ($3DØ-$3FF)
165 Table B-13. DOS Zero Page Use

Appendix C DOS Error Messages

168 Table C-1. DOS Error Messages
168 Table C-2. DOS Error Codes
170 Table C-3. Minimum and Maximum Values of Arguments
174 Table C-4. Types of Files According to Command

Preface

The *DOS Programmer's Manual* is written for people who want to store programs and other information on disks. It describes DOS 3.3, a disk operating system that allows you to place information on disks, change information already on disks, and retrieve information from disks. DOS runs on Apple II, Apple II Plus, and Apple IIe computers.

The *DOS User's Manual* can be used as an introduction to this manual. However, the *DOS Programmer's Manual* summarizes the information it contains.

Organization of This Manual

This manual is designed as a reference. The topics covered become increasingly complex as you progress.

Each chapter begins with an introduction to the topics it contains and concludes with a summary of the commands that are discussed.

Chapter 1 describes the DOS environment and the DOS commands that are used primarily from the keyboard.

Chapter 2 explains several ways of using DOS commands. It describes how to embed a DOS command in a program, monitor the processing, and intercept errors.

Chapter 3 tells you about disk access for text files and compares sequential and random access. It explains how to write sequential-access text programs.

Chapter 4 explains random-access files and how to write programs that use them.

Chapter 5 tells you how to use DOS to write a program that runs other programs, how to chain programs, and how to create a turnkey program.

Chapter 6 describes binary files and commands. It also discusses the relation between DOS and the Monitor program.

Appendix A describes how to run a program on a 13-sector disk and how to convert 13-sector disks to 16-sector format.

Appendix B explains disk storage: how tracks and sectors are mapped and used by DOS, and how machine-language programmers can directly address tracks and sectors.

Appendix C discusses DOS error messages. It lists each message and suggests ways to fix the problem.

Appendix D describes each program on the SYSTEM MASTER and the SAMPLE PROGRAMS disks.

Appendix E is a summary of the DOS commands.

The glossary explains some of the terms that are used in this manual. Terms shown in boldface in the text are defined in the glossary.

A tear-out reference card inside the back cover lists all the DOS commands.

Introduction

3 1.1 The DOS Environment
3 1.1.1 The DOS Disks
4 1.1.2 Requirements for Using DOS
4 1.1.3 A Few Conventions
5 1.2 How DOS Commands Are Described
5 1.2.1 Notation
5 1.2.2 Syntax
6 1.2.2.1 Defaults
6 1.2.2.2 Numbers
6 1.2.2.3 The Slot Number Option: [,Sn]
6 1.2.2.4 The Drive Number Option: [,Dn]
7 1.2.2.5 The Volume Number Option: [,Vn]
7 1.2.2.6 Hexadecimal Numbers
7 1.3 Direct DOS Commands
7 1.3.1 The CATALOG Command
8 1.3.2 The INIT Command
9 1.3.3 The RENAME Command
10 1.3.4 The LOCK Command
11 1.3.5 The UNLOCK Command
11 1.3.6 The DELETE Command
12 1.3.7 The VERIFY Command
13 1.3.8 The RUN Command
14 1.3.9 The LOAD Command
15 1.3.10 The SAVE Command

Introduction

1.1 The DOS Environment

The *DOS Programmer's Manual* describes how to use DOS commands to write and store programs and other information on disks. DOS is the disk operating system for the Apple II, Apple II Plus, and the Apple IIe computers.

When you bring the DOS program into the computer, the commands and capabilities of DOS are added to the other computing capabilities that are already available.

Here are some suggestions on how best to use the information in your DOS manuals.

- If you are just beginning to use an Apple II computer, you should be familiar with the introductory material that came with it.
- Before you begin using disks, it would help you to know the programming principles described in the *Applesoft Tutorial* and the *Applesoft BASIC Programmer's Reference Manual*.
- Depending on your familiarity with your computer and with programming techniques, you may want to read the *DOS User's Manual* before you begin to write programs that access disks. The *DOS User's Manual* is an introduction to the more complex information in this manual.

This chapter briefly summarizes information in the *DOS User's Manual*.

By the Way: There's a handy DOS summary card at the back of this manual. You can tear it out and keep it near your computer.

1.1.1 The DOS Disks

The disk labeled DOS 3.3 SYSTEM MASTER contains the DOS program itself and other programs that do additional tasks related to using disks. Many of these programs are described in this manual.

The disk labeled DOS 3.3 SAMPLE PROGRAMS contains demonstration programs that you can run while you are learning the DOS concepts. It also contains examples of BASIC programs that will help you learn to write programs that access disks. This manual describes many of these demonstration programs and all of the examples.

1.1.2 Requirements for Using DOS

The DOS operating system runs on any Apple II computer that has 16K (16 kilobytes) of **random-access memory** (RAM). DOS itself uses about 10.5K of memory. You'll find it easier to run your programs if your computer has at least 32K of memory. If you want to use both the Applesoft and the Integer BASIC programming languages, your computer must have 64K of memory.

Your Apple II computer must also have at least one disk drive connected to it. To connect the disk drive, follow the installation instructions in the manual that came with your disk drive.

DOS, version 3.3, organizes the space on a disk into 16 sectors.

Appendix A explains how to use a 13-sector disk with this 16-sector DOS.

1.1.3 A Few Conventions

Apple II, as used in this manual, implies every model of Apple II computer: the standard Apple II, the Apple II Plus, and the Apple IIe. The manual says explicitly when information applies only to a specific model.

The first time important terms are used, they appear in **boldface** type and are defined. These terms are also listed in the glossary.

The DOS manuals use two special paragraphs to catch your eye and deliver especially important information:

By the Way: The gray box provides a reminder or additional clarification—a faster or better way to do something. It is labeled "Hint" or "By the Way" or something similar.

Warning

The warning box indicates danger to a file, a disk, or your system. The information in the box describes the danger and suggests ways to avoid it.

If you have already read the *DOS User's Manual*, you can skip the material in the rest of this chapter. It is a summary, included here so you have all the information about DOS in one book.

1.2 How DOS Commands Are Described

The syntax (the order and form) of the various parts of a DOS command is expressed in a kind of shorthand, which is described in the next section.

1.2.1 Notation

This is the notation that is used in descriptions of command syntax:

UPPERCASE	indicates the actual name of something, like a DOS command. Type it exactly as indicated.
lowercase	indicates something you supply, like the name of a program.
fn	indicates a file name that you supply. A file name is composed of up to 30 characters. The first character must be a letter; the others can be any character, including a space, except the comma (,).
[ ]	Square brackets enclose an optional argument to a command. If you choose to include the argument, do not type the brackets, which are used only to indicate an option.
n	indicates a number you supply.

1.2.2 Syntax

All the possible forms of each DOS command are presented in a one-line description of the command (Figure 1-1).

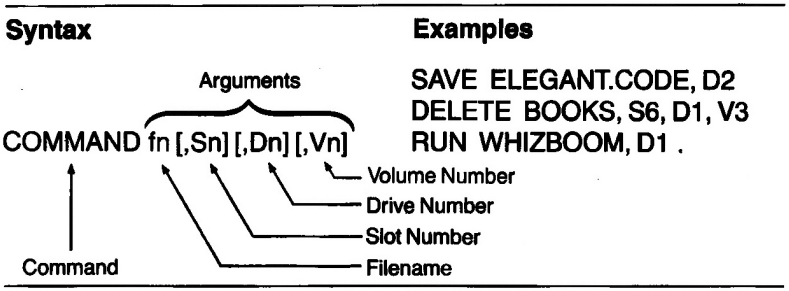


Figure 1-1. The Syntax of DOS Commands

In Figure 1-1, the word COMMAND represents any DOS command (for example, SAVE). The fn, [,Sn], [,Dn], and [,Vn] are the command's arguments. An argument in square brackets is optional; if omitted, DOS uses the default.

1.2.2.1 Defaults

A default is the value DOS assumes when you do not specify a value.

When you don't tell DOS anything about an optional argument, DOS makes an assumption called a **default**. A default value is what DOS uses when you do not explicitly give a value.

For example, suppose you type these commands in this order:

```
CATALOG,D1,S6
RUN  COPYA,D2                    (command without slot number)
```

The slot you specified in the first command, slot 6, becomes the default slot. When you omit the slot number in the RUN command, DOS uses the default, slot 6.

When you want to change the default, you must tell DOS explicitly; include the argument in the next appropriate command.

1.2.2.2 Numbers

Hexadecimal number system: base 16, digits 0 through 9 and A through F.

Replace the lowercase n in an argument with a number. You can use either a decimal integer or a **hexadecimal** number.

1.2.2.3 The Slot Number Option: [,Sn]

The Sn option specifies the number of the slot containing the disk controller card for the drive you want to use. Replace the lowercase n with an actual number from 1 to 7.

> **By the Way**: On Apple II and Apple II Plus computers, slot 0 holds the Language Card.

When you use the Sn option, the value you specify becomes the default slot number. That is, DOS uses this slot number until you specify a different slot number.

To boot = to start up

If you don't specify a slot number, DOS looks in the most recently used slot. If you never specify a slot number in a given session with DOS, DOS uses the slot from which you started (**booted**) DOS.

1.2.2.4 The Drive Number Option: [,Dn]

The Dn option specifies the number of the drive that contains the disk you want to use. Replace the lowercase n with a drive number, 1 or 2. When you omit the drive number, DOS uses the drive that it used last. If you never specify a drive number in a given session with DOS, DOS uses the drive from which you started DOS.

1.2.2.5 The Volume Number Option: [,Vn]

The Vn option specifies the volume number of the disk you want to use. Although the volume option is available with most DOS commands, it is rarely used. When you use Vn, replace the lowercase n with a volume number from 0 to 254; that number becomes the default volume number.

If you omit the volume number when you prepare a disk with the INIT command, DOS assigns a default number of 254 to that disk. In all other DOS commands, DOS ignores the volume specification when you omit this option, specify V0, or type V with no number.

1.2.2.6 Hexadecimal Numbers

Under certain circumstances, you can replace the lowercase n with a hexadecimal number. Machine-language programmers often use hexadecimal numbers because it takes fewer hexadecimal than decimal digits to express a large number.

You are never required to use a hexadecimal number in an option. But if you choose to, use a dollar sign ($) as the first character, followed by the hexadecimal digits. For example, the decimal integer 254 expressed in hexadecimal notation is $FE.

Quick Identification: Throughout this manual, hexadecimal numbers are preceded by a $ sign.

1.3 Direct DOS Commands

Warning
DOS commands must be typed in capital letters. If you are using an Apple IIe, press CAPS LOCK and leave it in the locked position.

1.3.1 The CATALOG Command

A catalog is a list of files on a volume.

The CATALOG command displays a directory of files called a **catalog**. The catalog lists the names, sizes, and types of files on a volume. Locked files are listed with an asterisk (*) next to their file type.

When a catalog contains more entries than your screen can display at one time, press RETURN to continue.

This is the command's full syntax:

CATALOG [,Sn] [,Dn]

[,Sn] where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.

[,Dn] where n is 1 or 2, specifies the disk drive to be accessed. If omitted, DOS uses the default drive number.

No Volume Number: If you include a volume number in a CATALOG command, DOS ignores it.

1.3.2 The INIT Command

The INIT command organizes the surface of a disk into tracks and sectors, writing zeros in all sectors. INIT also puts a BASIC file and a copy of DOS on the disk. This process creates an **initialized disk**. The DOS program on an initialized disk is always read into the same memory location.

An initialized disk is one prepared for use on an Apple II computer.

The INIT command takes this form:

INIT fn [,Sn] [,Dn] [,Vn]

Turnkey program = greeting program

fn is a name of the file to serve as the greeting or **turnkey** program.

[,Sn] where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.

[,Dn] where n is 1 or 2, specifies the drive that holds the disk on which to store the greeting or turnkey program. If omitted, DOS uses the default drive number.

[,Vn] where n is a number from 0 to 254, specifies a volume number to assign to a disk. If you specify 0, just V, or omit the option, DOS uses 254 as the default volume number.

1.3.3 The RENAME Command

The RENAME command changes the name of a file from the name indicated by fn1 to the name indicated by fn2. This is the form of the command:

RENAME fn1,fn2 [,Sn] [,Dn] [,Vn]

fn1,fn2 — where fn1 is the name of an existing file and fn2 is the new name that must be unique. If fn1 does not exist, you'll see a `FILE NOT FOUND` error message. The file must be unlocked.

[,Sn] — where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.

[,Dn] — where n is 1 or 2, specifies the drive that holds the disk containing the file to be renamed. If omitted, DOS uses the default drive number.

[,Vn] — where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS ignores the volume specification.

Warning

If fn2 existed before you executed RENAME, you may no longer be able to access the original contents of fn2 directly.

1.3.4 The LOCK Command

The LOCK command locks a file, that is, it protects an individual file from being accidentally altered, deleted, or renamed.

LOCK fn [,Sn] [,Dn] [,Vn]

fn	is the name of the file to be locked.
[,Sn]	where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.
[,Dn]	where n is 1 or 2, specifies the drive that holds the disk containing fn. If omitted, DOS uses the default drive number.
[,Vn]	where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS ignores the volume specification.

You'll see `FILE LOCKED` if you try to change, rename, or delete a locked file. To change a locked file, you must first unlock it by using the UNLOCK command.

In the catalog of the files on a disk, a locked file has an asterisk (*) to the left of its file type.

By the Way: You cannot lock a disk as a whole. You can, however, protect an entire disk by covering its write-enable notch with a write-protect tab.

1.3.5 The UNLOCK Command

The UNLOCK command unlocks a file, removing the file protection so that you can delete, rename, or change the file.

UNLOCK fn [,Sn] [,Dn] [,Vn]

fn	is the name of the file to unlock.
[,Sn]	where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.
[,Dn]	where n is 1 or 2, specifies the drive that holds the disk containing fn. If omitted, DOS uses the default drive number.
[,Vn]	where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS ignores the volume specification.

1.3.6 The DELETE Command

The DELETE command lets you remove a file from a disk.

DELETE fn [,Sn] [,Dn] [,Vn]

fn	is the name of the file to be deleted; fn must be unlocked. If the file does not exist, you'll see `FILE NOT FOUND`.
[,Sn]	where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.
[,Dn]	where n is 1 or 2, specifies the drive of the disk containing fn. If omitted, DOS uses the default drive number.
[,Vn]	where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS ignores the volume specification.

1.3.7 The VERIFY Command

The VERIFY command lets you test that a file was written on the disk correctly and that DOS can still read it.

VERIFY fn [,Sn] [,Dn] [,Vn]

fn — specifies the file you want to verify. Any type of file can be verified, including text files and binary files. DOS displays `FILE NOT FOUND` if the file does not exist.

[,Sn] — where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.

[,Dn] — where n is 1 or 2, specifies the drive that holds the disk containing fn. If omitted, DOS uses the default drive number.

[,Vn] — where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS ignores the volume specification.

If the file can be verified, it's safe to assume that the information on the disk has been stored correctly and can be retrieved whenever you want.

For more on buffers, see Chapter 5.

In verifying a file, DOS simply reads the file from the disk into a file buffer, an area in memory that is not currently in use. (This does not destroy a program that might already be in memory.) If DOS can read the file successfully, it displays the prompt character. If DOS finds that it cannot read the file (the file was damaged or written incorrectly), it displays the message `I/O ERROR`.

Silence Is Golden: When DOS can read the file, DOS does not display any message.

1.3.8 The RUN Command

See Section 2.2, BASIC Program Files.

The RUN command executes an Applesoft or Integer BASIC program that is stored on a disk.

RUN fn [,Sn] [,Dn] [,Vn]

fn — indicates the program that DOS is to run. Its file type must be either A or I. If the program is not on the disk, you'll see `FILE NOT FOUND`. If the file type is neither A nor I, you'll see `FILE TYPE MISMATCH`. If DOS cannot switch to the program's language, you'll see `LANGUAGE NOT AVAILABLE`.

[,Sn] — where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.

[,Dn] — where n is 1 or 2, specifies the drive that holds the disk containing fn. If omitted, DOS uses the default drive number.

[,Vn] — where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS ignores the volume specification.

When DOS sees the RUN command, it finds the specified program, brings it into memory, and starts it. Before bringing the program into memory, DOS checks the program's file type. If the file type is A and Applesoft BASIC is not active, DOS switches to Applesoft (if possible). If the file type is I and Integer BASIC is not active, DOS switches to Integer (if possible).

Since RUN automatically loads a program into memory, it is not necessary to use the LOAD command before you run a program.

By the Way: Once your program is in memory, you can run it again by issuing the RUN command without a file name. Without a file name, RUN is a BASIC statement.

1.3.9 The LOAD Command

The LOAD command transfers a copy of a disk program into memory.

LOAD fn [,Sn] [,Dn] [,Vn]

fn — indicates the program that DOS is to load. Its file type must be either A or I. If the program is not on the disk, you'll see `FILE NOT FOUND`. If the file type is neither A nor I, you'll see `FILE TYPE MISMATCH`. If DOS cannot switch to the program's language, you'll see `LANGUAGE NOT AVAILABLE`.

[,Sn] — where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.

[,Dn] — where n is 1 or 2, specifies the drive that holds the disk containing fn. If omitted, DOS uses the default drive number.

[,Vn] — where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS ignores the volume specification.

See Section 2.2, BASIC Program Files.

Before bringing the program into memory, DOS checks the program's file type and tries to switch to the appropriate BASIC. When you use the LOAD command to bring the contents of a file into memory, the file on the disk remains unchanged.

It is not necessary to load a program before you use the RUN command; RUN automatically loads it into memory.

By the Way: When a new program is loaded into memory, the previous program is erased from the Apple's memory. If you don't want to lose the program that is currently in memory, you must store it on a disk using the SAVE command.

1.3.10 The SAVE Command

See Section 2.2, BASIC Program Files.

The SAVE command transfers a copy of the BASIC program that is currently in memory to a file on a disk.

SAVE fn [,Sn] [,Dn] [,Vn]

fn	specifies the file name for the program. If a file by that name already exists on the disk, the file must be unlocked and must have the same file type (A or I) as the program you want to save.
[,Sn]	where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.
[,Dn]	where n is 1 or 2, specifies the drive that holds the disk containing fn. If omitted, DOS uses the default drive number.
[,Vn]	where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS ignores the volume specification.

When the current prompt is], DOS writes the file onto the disk with the file type A. When the prompt is >, DOS writes the file with the file type I. In addition to handling file type for you, DOS automatically determines the location on the disk and the length of the file being saved.

Warning

If you see `I/O ERROR` when you try to save a file, the disk may be bad or not initialized, the drive you specified may not contain a disk, or the drive door may be open. Put a fresh, initialized disk in the drive, close the door properly, and issue the SAVE command again.

DOS Programming Tools

19 2.1 Ways of Using DOS Commands
19 2.1.1 Immediate Execution
20 2.1.2 Deferred Execution
20 2.1.3 Command Files
20 2.2 BASIC Program Files
22 2.2.1 Switching Languages With the FP and INT Commands
23 2.2.1.1 The FP Command
23 2.2.1.2 The INT Command
24 2.2.2 Returning to BASIC From the Monitor Program
25 2.3 Using DOS From a Program
26 2.3.1 Example
27 2.3.2 Things to Watch Out For
28 2.4 Debugging With the MON and NOMON Commands
29 2.4.1 The MON Command
30 2.4.2 The NOMON Command
30 2.5 Intercepting Errors With Applesoft
31 2.5.1 Example
33 2.5.2 Finding Hidden Characters in a File Name
34 2.6 Talking to Other Devices From a Program
34 2.6.1 Starting a Program With PR# and IN#
35 2.6.2 The PR# Command
36 2.6.3 The IN# Command
36 2.7 The RENUMBER Program
37 2.7.1 RENUMBER Commands
38 2.7.1.1 The Syntax of RENUMBER Commands
39 2.7.1.2 Internal Line References
39 2.7.1.3 The Hold Buffer
40 2.7.1.4 Examples
41 2.7.2 RENUMBER Error Messages
43 2.8 Summary
43 2.8.1 Commands
43 2.8.2 Program

DOS Programming Tools

This chapter discusses how to use DOS commands from the keyboard and from a program. It describes:

- switching between the two types of BASIC and when to use each.
- how to monitor the action of the commands you have written.
- how to find errors in programs and hidden characters in file names.
- how your program can communicate with peripheral devices through the expansion slots.
- how to renumber your program instructions.

This chapter and those that follow describe more advanced ways of using DOS. The information is for people who are familiar with writing BASIC programs. If you have done all the examples in the *Applesoft Tutorial*, you know enough fundamental programming to use this manual.

2.1 Ways of Using DOS Commands

You can issue a DOS command directly from your keyboard, by embedding it in a BASIC program, or by including it in a command file.

2.1.1 Immediate Execution

Immediate execution: Commands issued from the keyboard.

You can issue most, but not all, DOS commands directly from your keyboard. This is called **immediate execution.**

Error messages; see Appendix C.

When you enter a command from the keyboard, DOS looks first at the command, then compares the command with its list of valid DOS commands and passes a command that is not in its list to BASIC. You will sometimes get error messages from BASIC when you meant to give DOS a command. In this case, check the proper form of the DOS command and issue it again.

2.1.2 Deferred Execution

Deferred-execution commands are executed when the program runs.

You can issue most DOS commands from within a BASIC program. This method of issuing commands is called **deferred execution** because commands embedded in a program are not executed until the program is run. For example, if a program contains the DOS command OPEN, the program will open a disk file to store information. The execution of the DOS command will be deferred until you issue the BASIC statement RUN to execute the program.

2.1.3 Command Files

EXEC file = command file.

A program that runs other programs is called a **command file**. Since the EXEC command runs a command file, a command file is also known as an **EXEC file**. The command file can contain both DOS commands and BASIC statements.

2.2 BASIC Program Files

An appendix in the *Applesoft BASIC Programmer's Reference Manual* summarizes the differences between Integer BASIC and Applesoft.

Apple II computers support two dialects of the **BASIC** (Beginners All-purpose Symbolic Instruction Code) programming language: **Applesoft BASIC** and **Integer BASIC**. Integer BASIC, the earlier version of the BASIC language, operates only on whole numbers (integers). Applesoft can operate on integers and on numbers expressed with decimal points (floating point). This is the most significant difference between them; for more details, see the *Applesoft BASIC Programmer's Reference Manual*.

On every Apple II computer, one of the two BASIC languages resides in **read-only memory** (ROM). This **resident language** is available to you as soon as you turn on your computer. In the standard Apple II, Integer BASIC is resident. In the Apple II Plus and Apple IIe, Applesoft BASIC is resident.

Add the alternate language by installing an Applesoft Card, an Integer Card, or a Language Card (also known as a 16K RAM Card).

The language you want to use depends on what work you want the computer to do for you. The language you *can* use depends on the memory size of your computer and the hardware or firmware it has. For example, if your Apple II computer has 48K or less of **random access memory** (RAM), you can have only one dialect in memory at the same time unless you add a card that supplies additional memory. It is relatively easy to add RAM memory cards.

Many Apple II computers are large enough to hold both dialects in memory. The Apple IIe computer is large enough to contain both dialects without additional hardware.

When you start DOS in an Apple II computer that has at least 64K of memory, DOS tries to load the **nonresident**, or alternate, BASIC from the disk into memory. DOS reports which BASIC it is loading into memory. On a computer with resident Applesoft BASIC, you'll see `LOADING INTEGER BASIC INTO MEMORY`. On a computer with resident Integer BASIC, you'll see `LOADING APPLESOFT BASIC INTO MEMORY`. On Apple II computers that have less than 48K of memory or when the alternate BASIC is not available on the startup disk, you will not see the `LOADING...` message.

You can recognize which BASIC is currently active by noticing the prompt character:

`]` indicates Applesoft BASIC.
`>` indicates Integer BASIC.

Use the FP or INT commands whenever you want to switch from one dialect to another.

Warning

The RUN and LOAD commands automatically try to activate the BASIC that corresponds to the program's file type.

On the Apple II Plus and Apple IIe computers, you'll see `LANGUAGE NOT AVAILABLE` when Integer BASIC, the nonresident language, is not already in memory.

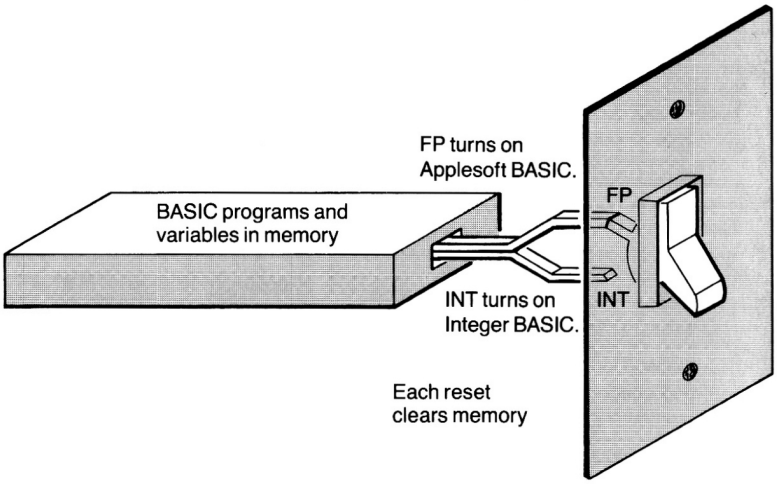
The APPLESOFT program loads the disk file FPBASIC, which contains Applesoft, into the Language Card.

On the standard Apple II computer, Integer BASIC is resident. When Applesoft is not in memory, DOS looks for the APPLESOFT program and runs it. If Applesoft BASIC is not on the disk, you'll see `LANGUAGE NOT AVAILABLE`.

2.2.1 Switching Languages With the FP and INT Commands

When you have both BASIC dialects in memory, use the FP and INT commands to switch from one to the other. You can also use FP or INT to reset the system and memory, even when you are returning to the same BASIC dialect. Both commands can be used in either immediate or deferred execution.

Figure 2-1. FP and INT



Warning

The FP and INT commands erase your program and variables from main memory by redirecting the pointers in memory. If you are going to switch languages, do so before typing in a new program.

2.2.1.1 The FP command

The FP command switches to Applesoft BASIC. FP stands for *floating point*. If you had been using Integer BASIC and then decided to write a program to balance your checkbook, you'd need to switch to Applesoft BASIC, which can operate on decimal numbers.

This is the form of the command:

FP [,Sn] [,Dn] [,Vn]

[,Sn]	where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.
[,Dn]	where n is 1 or 2, specifies the drive of the disk to be accessed. If omitted, DOS uses the default drive number.
[,Vn]	where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS ignores the volume specification.

Warning

Do *not* use the command RUN APPLESOFT to change languages. If you do, DOS will test the file type and set memory pointers to accommodate an Integer BASIC program instead of the Applesoft program you intend to work with.

See Chapter 6, Memory Usage and Entry Points.

2.2.1.2 The INT Command

The INT command switches to Integer BASIC. INT stands for *integer*. This is the full syntax of the command:

INT

The INT command does not have any arguments.

2.2.2 Returning to BASIC From the Monitor Program

For more about the Monitor program, see the *Apple II Reference Manual*.

Occasionally, you may see an asterisk (*), the prompting character of the Monitor program. Perhaps you were running a program that had an error in it and its error routine jumped to the Monitor.

You can return to BASIC from the Monitor in three ways. Two allow you to get right back to work; the third destroys what you had in memory, so you'll have to begin again.

1. To restart DOS and return to BASIC with your program and variables intact, type

   3D0G (careful, that's a zero...)

   It's No Dog: 3D0 is the address in memory where DOS starts. The G stands for *go*.

2. To return to BASIC with your program and variables intact, type CONTROL-C and press RETURN.

Warning

CONTROL-C only starts up the BASIC you were using before you entered the Monitor. If DOS was disconnected sometime prior to CONTROL-C, it will still be inactive even though you have returned to BASIC.

3. These commands also return to BASIC:

   CONTROL-B or 3D3G

Warning

CONTROL-B and 3D3G erase both your program and your variables from memory.

2.3 Using DOS From a Program

Using a DOS command from a BASIC program extends the capability of your programs in several ways. For example, you might use deferred-execution DOS commands to automatically display a disk's catalog, to save a backup copy of records in a file, or to save an unfinished game so you can continue it later.

To use a DOS command from a program, use the BASIC statement PRINT followed by the string you want to print. In this case, the string will contain a DOS command. To indicate that you're printing a DOS command rather than text, type a CONTROL-D as the first character of the string. This is the general form:

If you omit the CONTROL-D, DOS assumes the string is part of a BASIC statement and passes it on to BASIC's command interpreter.

PRINT CONTROL-D "DOS command"

You can use two methods to get a CONTROL-D into your program. In one method, the CONTROL-D is visible in the PRINT statement; in the other, the CONTROL-D is not visible.

To use an invisible CONTROL-D, type CONTROL-D right after you type the quote marks that begin the string you want to print:

```
10 PRINT "CATALOG"
```

(There is a CONTROL-D between "and C.)

The CONTROL-D is there, although you can't see it. This method can be used for both Integer BASIC or Applesoft program statements.

> **Caution**: Using the → key to copy a BASIC statement will *not* copy invisible control characters. In effect, the → deletes control characters.

A function returns a value for use in some further calculation.

You can see a CONTROL-D with Applesoft BASIC only. This method uses the **CHR$ function**. The CHR$ function takes a single numeric argument and returns a one-character string consisting of the corresponding ASCII character.

The **ASCII** (American Standard Code for Information Interchange) code for the CONTROL-D is 4. When you give CHR$ a value of 4, it returns a CONTROL-D. Set any string variable to CHR$(4) at the beginning of your program and print that string variable before each DOS command. You will able to see the variable in each statement. In addition, you'll need to fix it in only one place if you've declared it improperly.

In a PRINT statement, CONTROL-D from a CHR$ function looks like this:

```
5 REM VALID ONLY WITH APPLESOFT
10 D$=CHR$ (4)
20 PRINT D$;"CATALOG"
```

D$ is the string variable used throughout this manual. The name of the variable is a reminder of the control character it contains.

You Can Omit the Semicolon: The semicolon after D$ is optional. You'll save typing time and memory space by omitting the semicolon when your program has many DOS commands in PRINT statements. Without a semicolon, line 20 looks like this:

```
20 PRINT D$"CATALOG"
```

2.3.1 Example

The following Applesoft HELLO program displays a message and the disk's catalog on the screen. Type NEW to clear the computer's memory of any previous program and enter this program.

```
5 REM APPLESOFT HELLO
10 D$ = CHR$ (4) : REM CONTROL-D
                          (D$ contains CONTROL-D)
20 PRINT "DOS TEST PROGRAMS"
30 PRINT " 26 NOVEMBER 1985"
                          (Print a title and a date)
40 PRINT D$;"CATALOG"     (Then list the catalog)
50 END
```

Now type RUN to see how it works.

The equivalent program written in Integer BASIC looks like this.

```
5 REM INTEGER HELLO
10 D$ = "" : REM CONTROL-D
                          (There's an invisible CONTROL-D
                          between the quotes)
20 PRINT "DOS TEST PROGRAMS"
30 PRINT " 26 NOVEMBER 1985"
                          (Print the title and the date)
40 PRINT D$;"CATALOG"     (Then list the disk's directory)
50 END
```

Look closely at line 10, the only one that's different. When you use this method to set the D$ variable to CONTROL-D, it's common practice to include the remark, REM CONTROL-D.

Heads Up: From now on, most of the examples in this manual will be in Applesoft.

2.3.2 Things to Watch Out For

There are three things you should watch out for while using DOS commands from Applesoft programs.

First, *be sure you have only one DOS command per PRINT statement.*

A semicolon at the end of a PRINT statement prevents BASIC from adding a carriage-return character after a printed string, variable, or literal.

Second, *be sure D$ is preceded by a carriage return.* If the statement before your DOS command ends with a semicolon, a TAB, or a comma, your DOS command will not go to the DOS command interpreter.

If a DOS command does not start on a new line, the command will be printed and not executed. In this example, the word CATALOG is printed, but it is not executed as a command:

```
30 D$ = CHR$ (4) : REM CONTROL-D
                          (D$ is CONTROL-D)
40 PRINT "AUTUMN ";       (Semicolon inhibits RETURN)
50 PRINT D$;"CATALOG"     (So this doesn't work)
```

Instead of displaying a disk's catalog, this program prints AUTUMN CATALOG.

▲ **Warning**

When your program unexpectedly prints a DOS command instead of executing it, look for a program statement that ends with a semicolon.

If you want to be sure you print a carriage return before each CONTROL-D, set your D$ variable to contain both a carriage return (ASCII 13) and a CONTROL-D (ASCII 4):

```
D$=CHR$(13)+CHR$(4)
```

When D$ is declared this way, an Applesoft statement like PRINT D$;"CATALOG" prints a carriage return before the CONTROL-D and ensures that the DOS command always starts on a new line.

Warning
D$ declared this way will cause problems when you write or append text files.

The third thing to watch for is that *some DOS commands work only in programs*. The DOS commands that can be issued *only* in deferred execution are APPEND, OPEN, POSITION, READ, and WRITE.

2.4 Debugging With the MON and NOMON Commands

Program errors are often called **bugs**, and the process of getting a program to run properly is called **debugging**. To debug a program, you may want to follow the exchange of information between the disk and the computer. Monitoring this action helps you track down problems.

To watch this exchange, use the MON command. With MON, you can watch commands like OPEN that control operations to the disk. You can watch the output to a disk from PRINT statements or the input coming from a disk, like the data brought in by a READ command.

When you've seen all that you want to see (and to make your program run faster again), turn off the display with the NOMON command.

Perfect Combination: To see program statement numbers as you're debugging, combine DOS's MON and NOMON commands with BASIC's TRACE and NOTRACE statements. See the *Applesoft BASIC Programmer's Reference Manual* for details.

When you use TRACE, be sure your D$ variable contains both a carriage return and a CONTROL-D. Define D$ as D$ = CHR$(13) + CHR$(4)

2.4.1 The MON Command

Normally, you don't see all disk commands or the information sent between the computer and a disk. To monitor this action, use the MON command. The syntax of the MON command is:

MON [C] [,I] [,O]

C	displays all disk commands.
I	displays input, that is, information being sent from the disk to the computer.
O	displays output, that is, information being sent from the computer to the disk.

The arguments stand for Commands, Input, and Output, respectively. They may appear in any order and in any combination, depending on the information you wish to monitor. The commas are optional, for example: MONICO.

Warning
At least one of the arguments must be present or MON is ignored.

MON remains in effect until you issue a NOMON, an FP, or INT command, or you restart DOS with a startup (boot) disk, the command 3DØG, or the command 3D3G.

2.4.2 The NOMON Command

The NOMON command turns off the display of disk commands and information sent between the computer and the disk. The syntax of the NOMON command is:

NOMON [C] [,I] [,O]

C	suppresses the display of all disk commands.
I	suppresses the display of the input information going from the disk to the computer.
O	suppresses the display of the output information going from the computer to the disk.

The arguments stand for Commands, Input, and Output, respectively. They may appear in any order and in any combination, depending on what monitoring you want to suppress. The commas are optional.

Warning
At least one of the arguments must be present, or NOMON is ignored.

You can issue a NOMON command so that it prints almost invisibly. Use

```
10 PRINT D$; "NOMON C,I,O": VTAB PEEK(37):
     CALL -868
```

where D$ contains CONTROL-D, VTAB PEEK(37) moves the cursor to the beginning of the line that contains NOMON C,I,O, and CALL -868 clears that line.

2.5 Intercepting Errors With Applesoft

Error messages are listed in Appendix C.

When DOS or Applesoft detect an error connected with disk usage, they normally stop the program containing the error, display an error message, and store a code number for the error in memory location 222.

You can create an Applesoft routine that prevents both DOS and Applesoft errors from stopping your program. The routine can examine memory and return the error number, which can be tested. Based on this number, your program can branch to a line that handles that specific kind of error. To create an error-handling routine, use the ONERR GOTO statement and the PEEK function.

- The ONERR GOTO statement lets you specify a statement number to which to transfer control when your program encounters an error. Your error-handling routine can start at the statement specified in the ONERR GOTO statement.

- The PEEK function lets your program examine the contents of a specified location in the computer's memory; the location is specified as a decimal value. To detect the code number of an error, examine memory location 222; use `PRINT PEEK (222)` to see the code number of the error on your screen. You can also use `PEEK (222)` to assign the error as a numeric variable. To detect the statement number that caused the error, use `PEEK (218) + PEEK (219) * 256`. Precede the expression with `PRINT` to see the statement number on your screen.

PEEK must be used to determine a variable. PRINT PEEK prints the variable on the screen.

2.5.1 Example

The program called ONERR DEMO shows you how to use the ONERR GOTO statement and the PEEK function to recover from a DOS error. The program returns an error number and reports the number of the statement that caused the error.

The RENAME command won't work on a locked file. When your program tries to rename a locked file, DOS returns `FILE LOCKED` and the error-code number, 10.

ONERR DEMO can rename a file whether the file is locked or not. When ONERR DEMO detects an error, it uses the ONERR GOTO statement and branches to an error routine. The error routine gives you a chance to unlock the file before the program tries to rename the file again. The routine also displays the error number and line number of the error.

```
5 REM ONERR DEMO
10 D$ = CHR$ (4)          (D$ contains CONTROL-D)
15 PRINT D$;"MON C"       (Watch disk commands)
20 ONERR GOTO 100         (Handle an error at line 100)
30 INPUT "FILE TO RENAME? ";F$
                          (Read a file name 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 30 reads into F$ the name of the file to be renamed. Line 40 reads the new name into N$. Line 50 has the RENAME command. When there is no error, the program renames the file and ends.

When there is an error, the ONERR statement in line 20 sends the execution to line 100 where errors are handled.

```
100 PRINT: IF PEEK (222) <> 10 THEN 200
```

Line 100 tests location 222. If location 222 contains 10 (the code for FILE LOCKED), execution falls through to line 110 where you get a chance to rename the file. If location 222 does not contain 10, some other error occurred and execution goes to line 200.

```
110 INPUT "FILE IS LOCKED. RENAME ANYWAY? (Y/N)
    ";Y$
120 IF Y$ <> "Y" THEN 160
                              (No, don't rename file)
130 PRINT D$;"UNLOCK ";F$
                              (Yes, unlock the file)
140 PRINT D$;"RENAME ";F$;", ";N$
                              (Rename it)
150 PRINT D$;"LOCK ";N$  (Lock it again)
160 PRINT D$;"NOMON C": END
                              (Suppress special display)
```

Line 130 unlocks the file, line 140 renames it, and line 150 locks it again under its new name.

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

See Appendix C for the complete DOS error codes and their meanings.

When your program has an error other than FILE LOCKED, lines 200 and 210 display the code number of the error and the statement number in which the error occurs.

To bring ONERR DEMO in from the SAMPLE PROGRAMS disk and try it on a locked file, type

```
RUN ONERR DEMO
```

2.5.2 Finding Hidden Characters in File Names

You can also use Applesoft BASIC to find hidden characters in a file name. When a file name contains control characters, you won't see them printed, but you need to type them to use or delete the file.

If you suspect that you accidentally introduced control characters into a file name, you can use this Applesoft program to find any hidden character except CONTROL-M (carriage return), ESC, CONTROL-H (←), or CONTROL-U (→).

```
10 DATA 201, 141, 240, 21, 201, 136
20 DATA 240, 17, 201, 128, 144, 13
30 DATA 201, 160, 176, 9, 72, 132
40 DATA 53, 56, 233, 64, 76, 249
50 DATA 253, 76, 240, 253
60 FOR I = 768 TO 768 + 27
70 READ V : POKE I,V : NEXT I
80 POKE 54,0 : POKE 55,3
90 CALL 1002
```

Type this program, save it, and run it. When you then issue a CATALOG command, control characters in the file names will be displayed as blinking characters.

To return to normal display on an Apple II and Apple II Plus computer, type PR#0.

On an Apple IIe computer, **when you have not turned on the 80-Column Text Card**, type PR#0 to return to the standard 40-column display. If you have turned on the 80-Column Text Card, type PR#3 to return to the 80-column display, or type PR#3 and ESC CONTROL-Q to return to 40-column display.

▲ **Warning**

Never issue a PR#0 on an Apple IIe when the 80-Column Text Card is turned on. Doing so yields unpredictable results.

2.6 Talking to Other Devices From a Program

In this section, you'll find out how to have a program communicate with a peripheral device—for example, a printer or a disk drive—that is connected to the computer through a card in an expansion slot.

Your Apple II computer usually sends characters to the display screen, the standard output device. And it usually reads characters from the keyboard, the standard input device. The PR# and IN# commands allow you to use other devices.

Warning
You cannot have more than one peripheral device active at a time.

With DOS in effect, the PR# and IN# commands can be used in immediate execution in the usual way (see your BASIC manuals). But when PR# and IN# are issued by lines in a program, they must be in PRINT statements preceded by a CONTROL-D. For example

```
20 PRINT D$; "PR# 1"
30 PRINT D$; "IN# 2"
```

Without a CONTROL-D, PR# and IN# are BASIC commands, not DOS commands.

When you omit CONTROL-D from the PR# and IN# commands in deferred execution, DOS partially disconnects and is unable to print or read characters properly.

To restore DOS's input and output to the standard devices, use the BASIC statement:

For more information, see Section 6.4.1.

```
CALL 1002
```

2.6.1 Starting a Program With PR# and IN#

The primary purpose of PR# and IN# is to direct output and input. You can also use the commands to start a program on a disk in the drive connected to the slot specified in the command.

When you use PR# or IN#, DOS tries to run a program in the ROM chip on the card in that slot. When the program is in ROM on a disk controller card, the disk controller automatically tries to read information from the disk. Since reading information from the disk usually means bringing in the greeting program and running it, this amounts to starting the Apple II.

2.6.2 The PR# Command

The PR# command specifies one of the computer's slots and the device connected to the slot as the destination for output characters.

The PR# command can also be used to run a program on a disk by specifying the slot that contains your disk controller card. The syntax of the PR# command is:

PR# n

where n is a number from 1 to 7. DOS sends characters to the device connected through the slot specified by the number. **The number sign (#) is part of the command and must be typed.**

In a program use the command:
PRINT D$; "PR# n"

For example, when your computer has a printer controller card installed in slot 1 and you want your program to send output to the printer, use the command:

```
10 D$=CHR$(4)
20 PRINT D$; "PR# 1"
```

To send output to the screen again, on a standard Apple II or an Apple II Plus computer, use PR#0.

On an Apple IIe computer, **when you have not turned on the 80-Column Text Card,** use PR#0 to return to a 40-column screen. If the 80-Column Text Card is operating, use PR#3 to return to an 80-column display, or use PR#3 followed by ESC CONTROL-Q to go to a 40-column display.

By the Way: PR# 0 is a special case. It tells DOS to send output to the screen; it does *not* activate slot 0 on a standard Apple II or an Apple II Plus computer.

2.6.3 The IN# Command

The IN# command specifies a device as the source for input characters. The IN# command can also be used to run a program on a disk. Specify the slot that contains your disk controller card.

The syntax of the IN# command is:

IN# n

where n is a number from 1 to 7. DOS reads characters from the device connected through the slot specified by the number. **The number sign (#) is part of the command and must be typed.**

In a program use the command: PRINT D$; "IN# n"

For example, if your Apple II computer has an external terminal connected through slot 4, and you want your program to read characters from that external terminal, use the command:

```
10 D$=CHR$(4)
20 PRINT D$; "IN# 4"
```

On all Apple II computers, use IN# 0 to read input from the keyboard again.

> **By the Way:** IN# 0 is a special case. It tells DOS to read input from the keyboard; it does *not* activate slot 0.

2.7 The RENUMBER Program

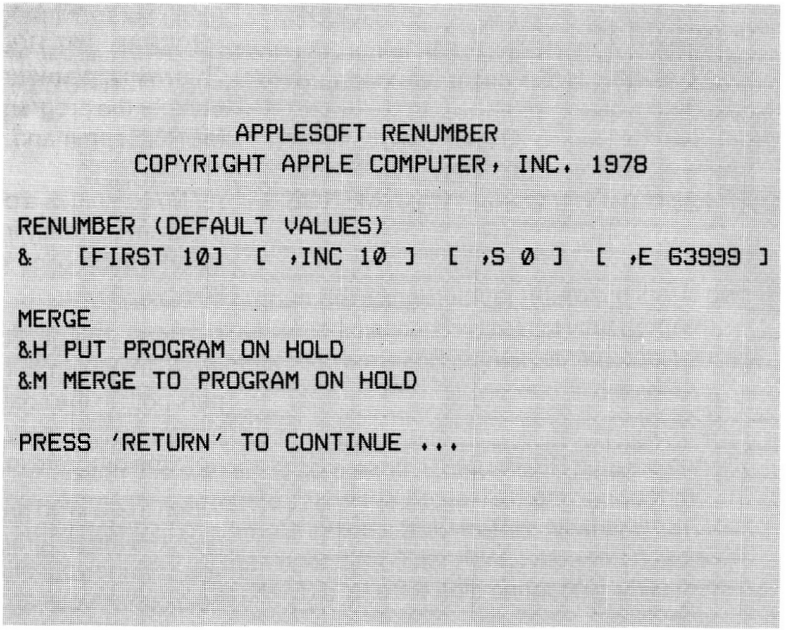
Use the RENUMBER program to renumber all or some of the statements of your Applesoft BASIC program, merge the statements of two of your programs, or insert a subroutine into your program from a subroutine library. The RENUMBER program resides on the SYSTEM MASTER disk. To use the program, put the SYSTEM MASTER disk in a drive and type

RENUMBER is a programming tool rather than a utility program.

```
RUN RENUMBER
```

After the RENUMBER title screen appears (Figure 2-2), issue one of the RENUMBER commands. You'll see a BASIC prompt character. RENUMBER stays in memory so you can continue to build your program by entering and changing BASIC statements. You can run and save your program, just as if the RENUMBER program were not there.

Figure 2-2. The RENUMBER Title Screen



2.7.1 RENUMBER Commands

The END, FIRST, INC, and START commands specify the renumbering values. Each value, n, must be a decimal number from 0 to 63,999.

START n (or Sn) where n is a line in the program that is currently in memory, specifies where to start to renumber. If you omit the S command, RENUMBER uses 0 (the first line in your program).

END n (or En) where n is a line number in the program that is currently in memory, specifies where to end renumbering. If you omit the E command, RENUMBER uses a value of 63,999.

FIRST n (or Fn) where n is a decimal integer, specifies the new number to assign to the starting line (the line specified by S). If you omit the F command, RENUMBER uses 10; that is, it renumbers the first line as 10.

INC n (or In) where n is the increment, specifies the size of the step to the next line number. If I is omitted, RENUMBER uses 10, to produce a sequence such as 110, 120, 130...

HOLD (or H) puts your program in the hold buffer and displays `PROGRAM ON HOLD, USE '&M' TO RECOVER.` This message means that whenever you want to transfer the program into memory again, issue the MERGE command.

MERGE (or M) combines a program in the hold buffer with the program currently in memory. The M command can insert a subroutine into your program from a subroutine library.

MERGE arranges the lines in ascending order. When there are statements with duplicate numbers, MERGE puts both in the final file, placing the statement from the program in memory first, followed by the statement from the program in the hold buffer.

When there is no program in memory, MERGE restores the program in the hold buffer to memory. If you change your mind after putting your program in the hold buffer, you can use the M command to restore the hold file as long as you do not load another program into memory.

When no program is in the hold buffer and you issue the M command, RENUMBER displays `NO PROGRAM IN MEMORY.`

2.7.1.1 The Syntax of RENUMBER Commands

The first character of a renumber command line is an ampersand (&). The shortest command line is: [&] [RETURN]. When you give this command, RENUMBER renumbers the program that is currently in memory, starting at the first and ending with the last program statement, by assigning 10 to the first program statement and incrementing the statement numbers by 10.

Use a comma to separate commands when there is more than one command on a line, for example, `&S 50, E 100, F 500` (which tells RENUMBER to process lines 50 through 100 of a program and give the first statement a line number of 500).

You may abbreviate a command to its first letter and the commands may be in any order.

2.7.1.2 Internal Line References

In addition to renumbering the statement numbers in your program, RENUMBER adjusts line-number references in these statements:

GOTO	ON...GOTO
GOSUB	ON...GOSUB
DEL	THEN (as in "7 IF X=0 THEN 250")
RUN	

RENUMBER will not renumber a line-number reference that is part of a remark. To have your remarks accurately reflect your program, renumber a line-number reference in REMARK statements yourself.

2.7.1.3 The Hold Buffer

When you use the HOLD command, RENUMBER puts the current program into a separate part of the Apple II's memory reserved for RENUMBER. This area is called a **hold buffer**. Neither BASIC nor DOS can use it for their operation. Therefore, when you need as much of memory as possible, use the hold buffer only while merging programs.

Warning

RENUMBER may take up to one minute to process a 16K program. During the time it is processing your program, do not press RESET! Always wait for the BASIC prompt before issuing the next command.

Pressing RESET or issuing the MERGE command while RENUMBER is running will destroy your program.

See Chapter 5, for details on command files.

Do not use the MAXFILES command while RENUMBER is in memory. DOS overwrites RENUMBER, destroying it.

RENUMBER commands may be put into a command file. However, when the program is executed (the EXEC command), RENUMBER returns to BASIC rather than to the command file.

RENUMBER executes CONVERT, HOLD, and MERGE immediately. If a command follows CONVERT, HOLD, or MERGE on the same line, RENUMBER ignores it.

2.7.1.4 Examples

The first example renumbers an entire program, the second renumbers part of a program, and the third merges two programs.

1. To renumber your entire program, starting at 10 and incrementing each instruction number by 10, put the SYSTEM MASTER disk in drive 1 and type

   ```
   RUN RENUMBER
   ```

   Put your program into memory by entering it from the keyboard or by using the LOAD command to read it into memory from a disk. After you see a BASIC prompt character, press the [&] and then [RETURN]. To see the renumbered statements, use the LIST command.

2. Here's a sample program that shows the renumbering part of a program. To try it, first be sure that RENUMBER is in memory, then type in the program listed on the left.

   Original Version

   ```
   1 INPUT X
   2 IF X<1 THEN 1
   3 ON X GOSUB 39,87
   27 END
   39 PRINT A
   45 RETURN
   87 PRINT A * A
   99 RETURN
   ```

   Renumbered Version

   ```
   1 INPUT X
   2 IF X<1 THEN 1
   3 ON X GOSUB 39,30
   27 END
   30 PRINT A * A
   39 PRINT A
   40 RETURN
   45 RETURN
   ```

   Now, give the command

   ```
   & START 87, END 99, FIRST 30
   ```

   which renumbers only two statements in the program. To see the renumbered program, issue the LIST command. Line 87 has become 30 and 99 has become 40. Note that RENUMBER has changed the line-number reference in line 3.

   Incidentally: This technique can be used when you want to move statements from one part of your program to another.

3. Suppose you want to merge two programs. With RENUMBER in memory, load your first program. You may add lines, renumber them, run the program to test it, and edit the program. When you are satisfied, put it into the hold buffer by typing

   ```
   & HOLD
   ```

   Now LOAD your second program. You can develop and test this program as you did the first. Renumber your second program so its lines don't conflict with the first program.

Finally, merge the programs. Type

```
& MERGE
```

You can, if you like, run the newly formed program to test it before you save it on the disk.

Sigh of Relief: If RENUMBER detects an irrecoverable error, it will stop execution before it makes any changes to your program. It displays an error message indicating the problem it found.

2.7.2 RENUMBER Error Messages

`> 63999`
: You entered a value that is out of range.

`DUPLICATE LINE NUMBERS`
: One of the new line numbers would be the same as a number you've specified if RENUMBER continues. Use a smaller increment (INCn) or a different number for first new line (Fn).

`HOLD FILE IN USE`
: You issued a second HOLD command without issuing an intervening MERGE command.

`LIMITED MEMORY, MAY DESTROY PROGRAM. CONTINUE (Y/N)?`
: You are operating in too small a system or with too large a program. If renumbering fails, your program may be destroyed. Any key other than Y cancels the renumbering and returns to BASIC.

`LINE INCREMENT = 0`
: Not allowed. Specify a value greater than 0.

`LINE INCREMENT TOO LARGE`
: The increment you specified would cause a line to be numbered beyond 63,999.

`LINE TOO LONG`
: Renumbering would cause a line longer that 239 characters. Use fewer statements per line.

`NO LINES IN RANGE`
: RENUMBER could not find any lines in the range you specified. It does not renumber any lines.

NO PROGRAM IN MEMORY

You tried to renumber with no program in memory. Return to memory the program that is in the hold buffer by typing &MERGE.

OUT OF MEMORY

Processing requires more memory than is available. RENUMBER itself is about 2K bytes long.

SYNTAX

RENUMBER does not recognize the first letter of a command or the value you specified is invalid.

2.8 Summary

2.8.1 Commands

FP [,S] [,D] [,V]

switches to Applesoft BASIC. It resets the pointers in memory so they no longer point to the previous BASIC program and variables.

INT

switches to Integer BASIC. It resets the pointers in memory so they no longer point to the previous BASIC program and variables.

MON [C] [,I] [,O]

displays the action between the computer and the disk.

NOMON [C] [,I] [,O]

suppresses the action display between the computer and the disk that was turned on by the MON command.

PR# n

specifies a device connected through slot n as the destination for output characters.

IN# n

specifies a device connected through slot n as the source of input characters.

2.8.2 Program

RENUMBER

renumbers the statements of all or a part of a BASIC program or merges the statements of two programs.

Using Sequential-Access Text Files

47 3.1 Comparing Sequential- and Random-Access Text Files
49 3.1.1 Choosing Sequential- or Random-Access Text Files
50 3.1.2 Current Position Pointer
50 3.2 Sequential-Access Text Files
50 3.2.1 The Field
51 3.2.1.1 Storing Characters in Fields
52 3.2.1.2 A Short Sequential-Access Text File
53 3.2.2 Entering and Reading Text
53 3.2.2.1 Writing To a File Using PRINT
54 3.2.2.2 Reading Characters From a File
55 3.2.2.3 One Part Per Field
57 3.2.2.4 Multiple Parts Per Field
59 3.2.2.5 Reading Fields That Contain Commas
61 3.2.3 Programs: Entering and Reading Text
61 3.2.3.1 A Program for Entering Text
63 3.2.3.2 A Program for Retrieving Text
65 3.3 Commands Used With Sequential-Access Text Files
65 3.3.1 The OPEN Command - Deferred Execution
66 3.3.2 The CLOSE Command - Immediate or Deferred Execution
67 3.3.3 The WRITE Command - Deferred Execution
68 3.3.4 The READ Command - Deferred Execution
68 3.3.5 The POSITION Command - Deferred Execution
69 3.3.6 The APPEND Command - Deferred Execution
71 3.4 Command Summary

Using Sequential-Access Text Files

For DOS commands used with random-access text files, see Chapter 4.

This chapter discusses text files and compares the two types: sequential-access and random-access. It then describes writing programs and the DOS commands used with sequential-access text files.

A text file is a series of fields stored on disk. A **field** is a sequence of from 1 to 32767 characters that ends with a carriage return.

A T in the file-type column of a catalog identifies both types of text files.

3.1 Comparing Sequential- and Random-Access Text Files

A sequential-access text file is a series of fields that can vary in length. Each successive field immediately follows the carriage-return character that ends the preceding field. Each time DOS writes to or reads from a sequential-access text file, DOS starts with the first field in the file and accesses the fields in sequence, one field after another.

A random-access text file is a series of **records**, each made of the same predetermined number of characters. Since a record can contain carriage-return characters, which indicate the end of a field, a record can contain one or more fields. But *all the records of a random-access text file are the same length*. Each time you open a random-access file, you must specify record length. This way, you can direct DOS to access any field in the file in any order.

Text files store strings of ASCII code (that is, text). However, the placement of the strings in the file depends on whether the text is written with sequential-access or random-access commands. Figure 3-1 shows how text is placed in a sequential-access file; Figure 3-2, in a random-access file (assuming a record length of 5 with one field per record).

The) character represents the RETURN character, which is sent automatically at the end of most PRINT statements.

Figure 3-1. Printing to a Sequential-Access Text File

Character: 7	)	A	T	)	O	N	E	)	B	L	O	W	)							
*ASCII:	B7	8D	C1	D4	8D	CF	CE	C5	8D	C2	CC	CF	D7	8D	00	00	00	00	00	00
File Byte:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Field:	0		1			2				3										

*The ASCII values are represented in hexadecimal numbers and reflect the fact that data is written to the disk with the high-order bit set.

Figure 3-2. Printing to a Random-Access File

Character: 7	)				A	T	)			O	N	E	)	B	L	O	W	)		
*ASCII:	B7	8D	00	00	00	C1	D4	8D	00	00	CF	CE	C5	8D	00	C2	CC	CF	D7	8D	00
File Byte:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Record Byte:	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0
Field:	0					0					0					0					
Record:	0					1					2					3					

*The ASCII values are represented in hexadecimal numbers and reflect the fact that data is written to the disk with the high-order bit set.

Note: The end of any text file is automatically marked with hexadecimal $ØØ.

Your program needs to open any text file before it tries to put anything into it. After you're finished working with the file, your program must close it. For these initial and final tasks, use the DOS commands OPEN and CLOSE.

When your text file is open, your program can print new lines to the file or read (retrieve) lines from it. Add information to a text file with the DOS command WRITE and the BASIC statement PRINT; retrieve information with the DOS command READ and the BASIC statements INPUT and GET.

See section 3.3 for the syntax and use of sequential-access commands.

See section 4.3 for the syntax and use of random-access commands.

The arguments you add to these commands and others depend on which of the two text-file types you are using. The main difference is that arguments to random-access commands specify length.

3.1.1 Choosing Sequential- or Random-Access Text Files

The fields of a sequential-access file can vary in length. You use them only sequentially, starting at the beginning of the file and working toward the end. Sequential-access is the better type of file for applications that begin a session by reading the entire contents of the file and finish by writing the modified contents back to the file. Many word processors store their text in sequential-access text files.

All the records in a random-access text file must be the same length. You can use them in any order or modify one record of the file without affecting the others. Random-access files are the better type for storing many pieces of information whose maximum length can be determined and whose contents change frequently. For example, you could store stock quotations or lists of experimental data in random-access files.

You might want to consider the following factors when choosing which type of text file to use:

- **Disk space.** The first time you write a record to a random-access text file, the entire record is placed in the file. If you specify a record length of 200 characters and write only one character to each record, you waste 199 (198 plus the carriage return) characters of disk space per record. In actual practice, records are rarely filled entirely, so random-access text files use disk space less efficiently than do sequential-access text files.
- **Amount of data.** When you read all the information into memory at the beginning of the program, it is faster to read it field by field from a sequential-access text file.
- **Use of data.** When 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.

3.1.2 Current Position Pointer

Every open text file has a pointer that keeps track of the **current position** in the file. Throughout this manual, this is called the **current position pointer**. When you read from a file, the current position becomes the character following the carriage return after the last character read. Likewise, when you write to a file, it points to the position immediately following the last carriage return. When you first open a file, the current position is the first character position in the file.

3.2 Sequential-Access Text Files

A sequential-access text file is like a scroll that contains an unlimited number of lines of text. With a sequential-access file, or a scroll, you need to search line by line to locate a particular text line. There are no pages to make the search faster.

3.2.1 The Field

The basic unit of a sequential-access text file is the **field**. A field can be compared to a line of text on the screen. Both are a series of characters that ends with a carriage-return character. To understand how to create a field in a text file, look first at how the PRINT statement of BASIC sends a field to the screen.

When a PRINT statement without a terminating semicolon prints a line to the screen, BASIC puts a carriage-return character at the end of the line. You can see the cursor move to the next line.

When you print to a file using a PRINT statement that does not have a terminating semicolon, BASIC puts the contents of the PRINT string into the file and writes a carriage-return character at the end of the string. In a file, the string is called a field. The next PRINT statement creates the next field in the file.

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

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

A semicolon tells BASIC not to write a carriage-return character at the end of a line (or a field). To make a longer line or field, use a semicolon after the string in the PRINT statement, for example

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

Because line 100 ends with a semicolon, the next PRINT statement will add characters to the same line on the screen (or to the same field in a text file):

```
110 PRINT " AND OTHER PARTS WILL FOLLOW."
```

On the screen, lines 100 and 110 look like this:

```
THIS MAKES PART OF A FIELD AND OTHER PARTS WILL
  FOLLOW.
```

3.2.1.1 Storing Characters in Fields

Look at the following PRINT statements: they create several fields in a file and show how the fields relate to each other in a sequential-access text file.

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

This sequential-access text file contains 36 characters in 6 fields of varying lengths. On the screen, lines 40 through 90 look like this:

```
GREEN
YELLOW
ORANGE
RED
VIOLET
BLUE
```

Here is how the characters generated by lines 40 through 90 are stored in a file:

The) symbol represents the carriage-return character.

```
Character sequence: GREEN)YELLOW)ORANGE)RED)VIOLET)BLUE)
Field number:       0     1      2      3   4       5
```

By the Way: Note that the first field in a sequential-access text file is numbered 0.

3.2.1.2 A Short Sequential-Access Text File

The sample program LISTSELF creates a sequential-access text file called LISTFILE and places lines in the file.

```
10 REM PROGRAM LISTSELF
20 D$ = CHR$ (4)          (D$ is CONTROL-D)
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 points there)
60 PRINT D$;"CLOSE"       (CLOSE all open files)
```

This short program directs DOS to open LISTFILE (line 30), uses WRITE so that LISTFILE can be written to (line 40), gives the BASIC statement LIST, and closes LISTFILE.

Notice that LIST is not a DOS command and is not preceded by a CONTROL-D. Indeed, if D$ were printed here, it would cancel the last DOS command (WRITE, in line 40).

The WRITE command redirects all output to the file. Therefore, the LIST statement places the lines of the program, one by one, into LISTFILE instead of sending them to the screen.

Type in the program. Then put an initialized, write-enabled disk into drive 1 and type

```
RUN
```

This executes LISTSELF, which puts the text file named LISTFILE on the disk. After LISTSELF has finished running and you see the prompt character on the screen again, look at the disk's catalog. You'll see the entry for LISTFILE. The two-sector long entry will be preceded by a T (for text file).

The EXEC command is described in Chapter 5.

To check the contents of the new text file, you can use the EXEC command. The EXEC command expects to deal with text files and to take commands from them instead of from the keyboard.

When you type in lines of a BASIC program, they are entered in memory as a BASIC program. Thus, if you use the EXEC command to read program lines from a sequential-access text file, they too will enter memory as a BASIC program. You can use the LIST statement to look at a BASIC program.

By the Way: Immediate-execution BASIC commands work on programs in memory whether you type them in, issue a LOAD command to bring them in from disk, or have an EXEC file bring them in.

The NEW statement clears memory and variables.

First, clear memory with the NEW statement. Then, issue the LIST statement to prove that there is no program in memory. Now type the command

```
EXEC LISTFILE
```

One prompt character appears on the screen for each line in the BASIC program. When the disk stops spinning and BASIC prints a prompt that is not followed by a line from the program, type

```
LIST
```

The LIST statement lists the program that has reappeared in memory.

3.2.2 Entering and Reading Text

How you enter text into a file determines how you will later read it from the file. Each of the following examples builds on the previous one.

In the next two discussions, the data separated by commas are called **parts**. Statement 5 in Table 3-1 shows how to create these parts; statement 2 in Table 3-2 shows how to read them.

3.2.2.1 Writing to a File Using PRINT

How you terminate the PRINT statement affects how characters are put into a text file. In Table 3-1, A$ has the value "DOG" and B$ has the value "CAT"; the) character denotes a carriage return.

Table 3-1. Printing to a Text File

PRINT Statement	Adds Characters	Comments
1. PRINT "TEXT"	TEXT)	Ends the current field; adds a carriage-return character.
2. PRINT "TEXT";	TEXT	Doesn't add a carriage return.
3. PRINT A$;B$;	DOGCAT	Adds second field to first field; doesn't add a carriage-return character.
4. PRINT A$,B$	DOGCAT)	Unlike PRINT to the screen, does not add spaces between fields separated by commas; adds a carriage return at the end.
5. PRINT A$;",";B$	DOG,CAT)	Adds a comma and second field to the first field; adds a carriage return.

3.2.2.2 Reading Characters From a File

How you put characters into a file with PRINT statements determines how you can read them. INPUT is better for reading some types of data; GET is better for others.

An INPUT statement contains a variable for each part of a field it can read. An INPUT statement with one variable reads all characters up to the next carriage-return character, adding to the variable only the characters up to a colon or comma. Since an INPUT statement can read data into more than one variable if the pieces of data are separated by commas, use INPUT with additional variables to read characters after a comma.

Use a GET statement to read information that cannot be read with an INPUT statement. GET reads all characters, including commas and colons. This is the way to read fields with varying numbers of parts.

Note: INPUT will truncate the information it reads to 239 characters. If the information is longer than 255 characters, INPUT will cancel the whole line and start over. This is why it is better to create many short fields than one long one. Also, it's a good idea to read from a sequential-access file with a GET statement when you're not sure of the length of the field you want to read.

Table 3-2 shows some of the ways to read characters with INPUT and GET.

Table 3-2. Reading From a Text File

Statement	Effect
1. INPUT A$	Reads one part of a field. If a field has more than one part, BASIC discards the remaining parts and displays `EXTRA IGNORED`. This means the other parts aren't read.
2. INPUT A$,B$	Reads two parts of a field. If a field has more than two parts, BASIC discards the remaining parts and displays `EXTRA IGNORED`. This means the other parts aren't read. If a field has less than two parts, BASIC reads parts from the next field.
3. GET C$	Reads the next character (letter, comma, or colon) from the file.

See the warnings about GET in section 3.2.2.5.

The next three examples show how to enter and read text with one part per field, with multiple parts per field, and with fields that contain commas.

3.2.2.3 One Part Per Field

This program shows you how to write four fields, each containing one part, to a sequential-access text file.

```
10 REM MAKE.FRUIT
20 D$ = CHR$ (4)          (D$ is CONTROL-D)
25 PRINT D$;"MON C,I,O"   (Watch the action)
30 PRINT D$;"OPEN FOUR.FRUITS"
                          (Create the file FOUR.FRUITS, if
                          necessary, and OPEN it)
40 PRINT "THIS GOES TO THE SCREEN"
                          (Characters go to the screen
                          until WRITE is executed)
50 PRINT D$;"WRITE FOUR.FRUITS"
                          (Specify the output file)
60 PRINT "APPLE"          (Put field 0 in the file)
70 PRINT "BANANA"         (Put field 1 in the file)
80 PRINT "CANTALOUPE"     (Put field 2 in the file)
90 PRINT "DATE"           (Put field 3 in the file)
92 PRINT D$;"CLOSE FOUR.FRUITS"
                          (Close FOUR.FRUITS)
94 PRINT D$;"NOMON C,I,O"
                          (Suppress the action)
96 END
```

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.

) = carriage-return character

```
Character sequence:  APPLE) BANANA) CANTALOUPE) DATE)
Field number:        0      1       2           3
```

The next program reads the four fields from FOUR.FRUITS with one input statement. It also displays them to show it's working.

```
10 REM GET.FRUIT WITH ONE INPUT STATEMENT
20 D$ = CHR$ (4)          (D$ is CONTROL-D)
25 PRINT D$;"MON C,I,O"   (Watch the action)
30 PRINT D$;"OPEN FOUR.FRUITS"
                          (Prepare FOUR.FRUITS for use)
40 PRINT D$;"READ FOUR.FRUITS"
                          (Specify file for input)
50 INPUT A$(0),A$(1),A$(2), A$(3)
                          (If there are no parts left in a
                          field, DOS reads the first part
                          from the next field)
80 PRINT A$(0)" "A$(1)" "A$(2)" "A$(3)
                          (Print the fruits on the screen)
90 PRINT D$;"CLOSE FOUR.FRUITS"
                          (Close FOUR.FRUITS when
                          done)
94 PRINT D$;"NOMON C,I,O"
                          (Suppress the action)
96 END
```

You'll see this on your screen:

```
APPLE BANANA CANTALOUPE DATE
```

An array is a table of elements (numbers or words).

The next program reads four fields of the sequential-access text file FOUR.FRUITS into successive elements of an **array**. The DIM statement in line 15 sets aside space for an array of the specified number of **elements**.

```
10 REM GET.FRUIT.INTO.CUPS USING AN ARRAY
15 DIM A$(3)
20 D$ = CHR$ (4)          (D$ is CONTROL-D)
25 PRINT D$;"MON C,I,O"   (Watch the action)
30 PRINT D$;"OPEN FOUR.FRUITS"
                          (Prepare FOUR.FRUITS for use)
40 PRINT D$;"READ FOUR.FRUITS"
                          (Prepare it for reading)
50 FOR I = 0 TO 3         (For fields 0 through 3 read field I
60 : INPUT A$(I)          from the file, and print it on the
70 : PRINT A$(I)          screen)
80 NEXT I                 (Then do the next field)
90 PRINT D$;"CLOSE FOUR.FRUITS"
                          (Close FOUR.FRUITS when
                          done)
```

```
94 PRINT D$;"NOMON C,I,O"
                                    (Suppress the action)
96 END
```

This program uses the INPUT statement (line 60) once for each field it reads from the file. The colons in lines 60 and 70 are not required; they make the program easier for you to read.

3.2.2.4 Multiple Parts Per Field

The next program places three parts in each of two fields. When you type the program into memory and run it, the commas between the parts in lines 50 and 60 are written to the file INVENTORY because they are within quotes.

```
10 REM BINS
20 D$ = CHR$ (4)              (D$ is CONTROL-D)
25 PRINT D$;"MON C,I,O"       (Watch the action)
30 PRINT D$;"OPEN INVENTORY"
                                    (Create the file INVENTORY, if
                                    necessary; prepare to use it)
40 PRINT D$;"WRITE INVENTORY"
                                    (Prepare to write to file)
50 PRINT "BOLTS,SCREWS,NAILS"
                                    (Print three parts in field 0)
60 PRINT "WASHERS,GROMMETS,RINGS"
                                    (Print three parts in field 1)
70 PRINT D$;"CLOSE INVENTORY"
                                    (Close INVENTORY)
94 PRINT D$;"NOMON C,I,O"
                                    (Suppress the action)
96 END
```

Here's how the characters are stored in INVENTORY:

```
Character
sequence:  BOLTS,SCREWS,NAILS)WASHERS,GROMMETS,RINGS)
Field      0                  1
number:
```

The INPUT statements in the next two programs will read these commas in different ways. The commas are treated as markers for the end of the parts the programs read (retrieve).

This program reads each part into a separate variable.

```
10 REM DRAWERS
20 D$ = CHR$ (4)              (D$ is CONTROL-D)
25 PRINT D$;"MON C,I,O"       (Watch the action)
30 PRINT D$;"OPEN INVENTORY"
                              (Prepare to use INVENTORY)
40 PRINT D$;"READ INVENTORY"
                              (Prepare to read from file)
50 INPUT A1$,A2$,A3$,A4$,A5$,A6$
                              (Read 3 parts from field 0 and 3
                              parts from field 1)
60 PRINT D$;"CLOSE INVENTORY"
                              (Close INVENTORY)
70 PRINT A1$,A2$,A3$,A4$,A5$,A6$
                              (Print all six parts)
80 PRINT D$;"NOMON C,I,O"
                              (Suppress the action)
90 END
```

Notice that line 50 simply reads consecutive parts from the file. When all the parts have been read from one field, parts are automatically taken from the next field.

This program reads only the first part from each field.

```
5 REM METAL.PARTS
10 D$ = CHR$ (4)              (D$ is CONTROL-D)
20 PRINT D$;"MON C,I,O"       (Watch the action)
30 PRINT D$;"OPEN INVENTORY"
                              (Prepare to use INVENTORY)
40 PRINT D$;"READ INVENTORY"
                              (Prepare to read from file)
50 INPUT A1$                  (Read first part from field 0)
60 INPUT A2$                  (Read first part from field 1)
70 PRINT D$;"CLOSE INVENTORY"
                              (Close INVENTORY)
80 PRINT A1$,A2$              (Display the two parts)
90 PRINT D$;"NOMON C,I,O"
                              (Suppress the action)
100 END
```

In this example, each INPUT statement reads an entire field, regardless of the number of parts it has. Line 50 assigns the value "BOLTS" to A1$, discards the rest of field 0, and displays EXTRA IGNORED. Likewise, line 60 assigns the value "WASHERS" to A2$, ignores the rest of field 1, and displays EXTRA IGNORED. Finally, line 70 displays the parts BOLTS and WASHERS.

3.2.2.5 *Reading Fields That Contain Commas*

To read a comma, colon, or control character, or to detect a particular character as it is read, use the Applesoft GET statement instead of INPUT. INPUT reads strings of characters separated by a comma, but GET reads characters one by one from either the keyboard or a text file.

Warning

After an Applesoft GET statement reads a character from a text file, the following problems arise in DOS:

If a DOS command is the first item printed after the GET, the DOS command may not be executed because the necessary preceding carriage-return character is missing. To solve this, print a carriage return before printing the DOS command.

When NOMON C,I,O is in effect, the first character printed after GET *will not appear* on the screen.

When MON C,I,O is in effect, the first character printed after GET *will* appear on the screen.

To solve these problems, put a nonprinting character (like CONTROL-A) into a variable and print it before the PRINT character you want to see.

Note also that GET reads a field character by character and stores each character it reads until it arrives at a carriage-return character. Thus, using GET to read a long string or a series of concatenated strings may cause memory problems.

If you've been doing the examples, you typed in the BINS program and created the file INVENTORY. The next example retrieves characters from INVENTORY by using the GET statement. The GET statement reads one or more parts from a field.

INVENTORY has three parts in each field. However, there may be times when you won't know how many parts to read from a field. Use the GET statement to read an unknown number of parts, separated by commas, from one field of a file.

The following subroutine reads parts, 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 subroutine also uses the GET statement to read one character into the variable C$. If the character is not a comma or a carriage-return character, the subroutine adds it to A$(I). When the subroutine reads a comma that separates two elements, it increments the variable I by 1, so that I will indicate the next element of the array, and continues reading characters. It repeats this process until it reads a carriage-return character indicating the end of the field.

```
10 R$ = CHR$ (13)          (R$ is a 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 GET reads a carriage return,
                           there are no more elements to
                           read; otherwise
1050 A$(I) = A$(I) + C$    add C$ to the element)
1060 GOTO 1020
```

To use this subroutine, you need a program that calls the subroutine to retrieve the parts from the first field of the file INVENTORY. To summon a subroutine, use GOSUB n, where n is the line on which the subroutine starts (line 60 below). When the RETURN statement is executed (line 1040 above), execution branches to the line following the GOSUB statement (line 70 below).

This example reads only the first field from the file.

```
20 REM USE SUBROUTINE
30 D$ = CHR$ (13) + CHR$ (4)
                           (D$ is carriage return plus
                           CONTROL-D to ensure that CLOSE
                           executes)
40 PRINT D$;"OPEN INVENTORY"
                           (Prepare INVENTORY for use)
50 PRINT D$;"READ INVENTORY"
                           (Prepare INVENTORY for
                           reading)
60 GOSUB 1000              (Read all parts from a field)
70 PRINT D$;"CLOSE INVENTORY"
                           (Close INVENTORY)
80 FOR J = 1 TO I          (Now print the I elements of A$
                           onto the screen)
90 : PRINT A$(J)
100 NEXT J
110 END
```

You can test this program. After you have typed in the lines of the program and the lines of the subroutine, type

```
RUN
```

You'll see the three words in the first field of the INVENTORY file on your screen, one word per line.

3.2.3 Programs: Entering and Reading Text

You can see how GET reads a variable number of fields from a file by using two programs from the SAMPLE PROGRAMS disk. The program MAKE TEXT reads text from the keyboard and saves it in a file. The program GET TEXT reads text from a file and displays it on the screen.

3.2.3.1 A Program for Entering Text

The following program is stored in the file MAKE TEXT. The program lets you type 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 when it encounters an empty line. The first portion of MAKE TEXT looks like this:

```
5 REM   MAKE TEXT
10 DIM A$(100)              (Allow room for 100 lines in
                            array)
20 D$=CHR$(4) : REM CONTROL-D
                            (Set CONTROL-D in the
                            variable D$)
30 R$=CHR$(13) : REM RETURN
                            (Set RETURN in the variable R$)
40 TEXT : HOME              (Set text mode; clear screen)
50 PRINT "      TEXT FILE CREATOR"
60 INVERSE : PRINT "      TO ADD A STRING:"
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       (Skip 1 line; specify line 6 as top
                            of screen for new display to save
                            program instructions)
```

The TEXT statement converts the display to 24 lines of text (instead of graphics) and positions the cursor at the beginning of the bottom line.

```
110 I = I + 1 : PRINT I;": ";
                                    (Ask for next line of text)
120 GOSUB 1000                      (Jump to reading routine)
130 IF A$(I) < > "" GOTO 110
                                    (If A$(I) is not empty, go to line
                                    110 for another line)
```

In line 40, the TEXT statement switches to text mode and the HOME statement clears all characters from the screen and moves the cursor to the upper-left corner of the screen. Lines 50 through 110 place the instructions for the program on the screen, some in normal letters, and some in inverse letters. Line 100 *sets a text window*, that is, it freezes the upper six lines of the screen so that the instructions stay on the screen. The variable I has the value of 0 the first time it is used. When the program executes line 110 the first time, it sets I to 1 and prints "1" on the screen.

Setting a text window, see the *Applesoft BASIC Programmer's Reference Manual*.

Line 120 directs MAKE TEXT to jump to the subroutine that reads a line of text from the keyboard. This is the subroutine:

```
1000 GET C$: PRINT C$;          (Read 1 character and print it to
                                the screen)
1010 IF C$=R$ THEN RETURN
                                (If that character = RETURN, exit this
                                routine; go back to 130)
1020 A$(I) = A$(I) + C$         (If something else was input,
                                build it into a string)
1030 GOTO 1000                  (Go back for more input)
```

MAKE TEXT reads one character into the variable C$ and tests C$. If C$ contains a carriage-return character, execution goes to line 130. But if C$ does not contain a carriage-return character, the program adds the contents of C$ into the growing array (line 1020). The program puts each string into element I of the array A$.

Once the text is stored in the array, MAKE TEXT asks for the name of the file into which it should place the text:

```
140 INPUT "SAVE TO WHAT FILE? ";N$
150 IF LEN (N$)=0 THEN 220
                                (If no name given, go to 220 to
                                quit)
160 PRINT D$;"OPEN ";N$         (Open the file named N$)
170 PRINT D$;"WRITE ";N$        (Prepare to write to it)
```

The program reads that name into the variable N$. The next line tests that a name was actually entered. If a file name was not entered, the program branches to line 220 and quits.

Note: If the filename does not begin with a letter, DOS will display SYNTAX ERROR.

Next, the program saves the array contents into the specified file. An empty input line indicates the end of the text. MAKE TEXT then prints the entire array to the file (lines 180 through 200).

```
180 FOR J = 1 TO I - 1    (For each line of text, it
190 : PRINT A$(J)         prints the line to the file
200 NEXT J                and continues to the next line)
210 PRINT D$;"CLOSE ";N$  (Closes the file when done)
```

Finally, with line 220, MAKE TEXT resets the screen pointers for proper text mode and ends.

```
220 TEXT : END             (Resets screen to normal size)
```

Run this program a few times, creating text files of different lengths. Experiment with the program's features and get familiar with the way they work. For example, enter a blank line of text by putting spaces on that line; since it contains characters, the length of the line will not be 0.

3.2.3.2 A Program for Retrieving Text

Creating a sequential-access text file is a challenging exercise. You also need to have some method for retrieving the information you've put into the file.

The program GET TEXT reads a sequential-access text file. Load the program from the SAMPLE PROGRAMS disk and look at the lines as they are described below.

First, GET TEXT sets up the variables it is going to use. It dimensions the array A$ to hold up to 100 elements and assigns the value of CONTROL-D to D$ and the value of RETURN to R$. Before going on, the program clears the screen and moves the cursor to the upper-left corner of the screen (line 50).

```
5 REM GET TEXT
10 ONERR GOTO 150
20 DIM A$(100)
30 R$ = CHR$ (13)          (R$ is RETURN)
40 D$ = CHR$ (4)           (D$ is CONTROL-D)
50 TEXT : HOME
```

In line 60, GET TEXT asks for the name of the file from which to read text and reads that name into the variable N$. Line 70 tells you that pressing (CONTROL)-(S) will stop the program and pressing almost any key will restart the listing.

```
60 INPUT "LIST WHAT TEXT FILE? ";N$
70 PRINT : PRINT "TYPE: CONTROL-S TO STOP
     LISTING"
80 PRINT "                ANY KEY TO CONTINUE"
90 PRINT : PRINT : POKE 34,6
```

As with the preceding program, GET TEXT freezes the upper six lines of the screen (line 90) so that they remain on the screen even when you enter more lines than the screen can hold.

Having read the name of the text file, GET TEXT can now open the file you've specified and read consecutive lines from it:

```
100 PRINT D$; "OPEN "; N$
                              (Open the text file named in N$)
110 PRINT D$; "READ "; N$
                              (Prepare to read from the file)
120 FOR I = 1 TO 100          (For each line in the file, go to
130 GOSUB 1000                the subroutine to process each
                              character)
135 : PRINT A$(I)             (Display each character on the
                              screen)
140 NEXT I                    (Then do the next string)
```

The only remaining task is to close the file and reset the screen.

```
150 PRINT D$;"CLOSE ";N$ (Close the file)
160 TEXT : END
```

Here is the subroutine mentioned in line 130. It is similar to the subroutine of MAKE TEXT.

```
1000 GET C$: PRINT C$;    (Read 1 character and echo it)
1010 IF C$=R$ THEN RETURN
                              (If that character was a (RETURN), exit
                              this routine and go back to 135)
1020 A$(I) = A$(I) + C$   (If something else was there,
                              build it into a string)
1030 GOTO 1000                (Read more)
```

3.3 Commands Used With Sequential-Access Files

This section describes the DOS commands used with sequential-access files. Note that only one of these commands can be used in both immediate and deferred execution. The others can be used only in deferred execution.

3.3.1 The OPEN Command - Deferred Execution

Before writing to or reading from a sequential-access text file, a program must open that file.

When a program opens a text file, DOS reserves 595 bytes of memory space for the file; this space is called a **file buffer**. DOS also sets the current position (for reading and writing) to point to the beginning of the file.

MAXFILES, see Chapter 5.

The OPEN command allows up to 16 files to be open at once; however, you must set MAXFILES since each DOS file you use requires a file buffer.

This is the general form of OPEN:

OPEN fn [,Sn] [,Dn] [,Vn]

fn — indicates the name of the file to be opened. When it does not yet exist, DOS creates one with that name, type T. If the file already exists, it must not be open.

[,Sn] — where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.

[,Dn] — where n is 1 or 2, specifies the drive that holds the disk containing fn. If omitted, DOS uses the default drive number.

[,Vn] — where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS ignores the volume specification.

Previous sections presented several examples of the OPEN command. This section gives you a new use for OPEN.

Suppose your program routinely replaces an old text file with a new one with the same name. When the new one is shorter than the old one, the new one will have part of the old file hanging on the end unless the program first deletes the old one. If you don't want this extra text at the end of the file, delete the old file before writing to the new one. The next program shows you how to delete a file and open it for new data, whether or not the file already exists.

```
5 REM DEMONSTRATES ERROR FREE DELETION
10 D$ = CHR$ (4)              (D$ is CONTROL-D)
20 N$ = "GAMES"               (N$ is the name of the file)
30 PRINT D$;"MON C,I,O"       (Watch the action)
40 PRINT D$;"OPEN ";N$        (Open the file. If it does not
                               already exist, DOS creates file)
50 PRINT D$;"DELETE ";N$      (Since the file definitely exists,
                               the file can be deleted)
60 PRINT D$;"OPEN ";N$        (Open the file)
```

The following statements complete the example program.

```
70 PRINT D$;"WRITE ";N$       (Prepare to write to the file)
80 PRINT "TIC"
90 PRINT "TAC"
100 PRINT "TOE"
110 PRINT D$;"CLOSE ";N$      (Close file)
120 PRINT D$;"OPEN ";N$       (Open the file)
130 PRINT D$;"READ ";N$       (Read the file)
140 INPUT A$,B$,C$
150 PRINT A$,B$,C$
160 PRINT D$;"CLOSE ";N$      (Close file)
170 PRINT D$;"NOMON C,I,O"
                               (Suppress the action)
180 END
```

By the Way: Deleting a text file frees that file's allocated file buffer as well as the file's sectors on the disk.

3.3.2 The CLOSE Command - Immediate or Deferred Execution

For more on file buffers, see Section 5.3.

After a program has finished writing to or reading from a file, it must close the file. Closing every file properly ensures that all characters are written to the files and that the file buffer is released.

By the way: A program should always close a file that it opened as soon as possible, even if only temporarily. You or your program can always open the file again if you need it.

In some circumstances, a program contains an error and stops before it can close all open files. When this happens, issue the CLOSE command from the keyboard.

CLOSE takes the form:

CLOSE [fn]

where fn indicates the name of the file to be closed. The CLOSE command without any arguments closes all open files.

A Coding Technique: Add an ONERR GOTO statement that branches to a routine that will close all files.

3.3.3 The WRITE Command - Deferred Execution

The WRITE command tells DOS which file to write to and where to start writing. The WRITE command remains in effect until the next DOS command or an INPUT statement. You must use the WRITE command before you can use the PRINT statement to place characters in a file.

After a WRITE statement, all output characters that would normally be displayed on the screen are sent to the file. This includes the question-mark prompting character of an INPUT statement (if INPUT appears after the WRITE command) and error messages (unless an ONERR GOTO statement redirects them).

This command takes the form:

WRITE fn [,Bn]

fn — indicates the name of the sequential-access text file to write to. This argument is required.

[,Bn] — where n is the number of bytes, or characters, to read and skip over. This option changes the current position in the file.

You can use the Bn option to tell DOS where in the file to write the first character. If you don't use it, DOS writes the first character at the current position.

By the Way: If you have not included an ONERR GOTO statement and a DOS error occurs, the error message will be stored as the last field in your text file. In addition, the error message will cancel a WRITE command.

3.3.4 The READ Command - Deferred Execution

The READ command tells DOS which file to read from. It remains in effect until the next DOS command is given. You must use the READ statement before you use the INPUT and GET statements to read characters from a file.

This command takes the form:

READ fn [,Bn]

fn — indicates the name of the file to read from. This argument is required.

[,Bn] — where n is the number of bytes, or characters, to read and skip over. This option changes the current position in the file.

You can use the Bn option to tell DOS where in the file to read the first character. If you don't use it, DOS reads the first character at the current position.

Reminder: Avoid using TRACE (an Applesoft statement) with INPUT statements unless you have defined your D$ variable to contain both a carriage return and a CONTROL-D: `D$=CHR$(13)+CHR$(4)`.

3.3.5 The POSITION Command - Deferred Execution

The POSITION command lets you access information in any field within a file. It automatically opens the specified file and directs DOS to start reading at the current position and to skip over a number of fields. Since a field is a sequence of characters ending with a carriage return, POSITION scans the file, byte by byte, looking for carriage-return characters.

Warning
The POSITION command cancels a READ or WRITE command; therefore, use POSITION before a READ or WRITE.

This command takes the form:

POSITION fn [,Rn]

fn — indicates the file whose current position is to be altered. This argument is required.

[,Rn] — where n indicates the number of fields to read and skip over. This number is relative to the current position. When Rn is omitted, it defaults to 0, the first field. If you try to specify a position past the end of the file, DOS displays `END OF DATA` and stops program execution.

This command specifies that the current position be moved forward to the Rth field ahead of the current position. When R is specified as 0, DOS reads or writes beginning in the current field. When R is specified as 1, DOS skips the current field and begins reading or writing in the next field.

The position specified is **relative** to the current position. For example, if the current position is in the fourth field of a file and you want to read from the tenth field in the file, skip six fields by using POSITION fn,R6.

Warning

If you use POSITION to rewrite the current field, make sure that you print exactly the same number of characters that you printed in that field originally. If you print fewer characters, you will create a new field: the field just printed and the end of the original field. If you print more characters than the original field contains, you will write over some of the characters of the next field in the file.

3.3.6 The APPEND Command - Deferred Execution

The APPEND command opens a sequential-access text file and sets the current position to the end of the file so that you may add information. After opening the file, the APPEND command sets the current position to point to one byte beyond the last character in the file. *The APPEND command must be followed by a WRITE command.*

The APPEND command has this form:

APPEND fn [,Sn] [,Dn] [,Vn]

fn	indicates the file to append to. When the file does not exist, DOS displays `FILE NOT FOUND`.
[,Sn]	where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.
[,Dn]	where n is 1 or 2, specifies the drive that holds the disk containing fn. If omitted, DOS uses the default drive number.
[,Vn]	where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS ignores the volume specification.

Reminder: Unlike the OPEN command, the APPEND command does not create a new file. The file you specify must already exist.

MAKE TEXT creates a sequential-access text file

The next example modifies the program MAKE TEXT so that it adds lines to the end of a text file. First MAKE TEXT reads lines of text into the array A$ and asks for a file name. Then it opens the file and prints the text lines to the file.

With the SAMPLE PROGRAMS disk in drive 1, load MAKE TEXT into memory and display it on the screen by typing

```
LOAD MAKE TEXT
LIST
```

Change the code to append text lines to the end of the file by replacing the OPEN statement with an APPEND statement. To replace lines 50 and 160, enter these lines

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

To keep this program, save it by typing

```
SAVE APPEND TEXT
```

3.4 Command Summary

OPEN fn [,Sn] [,Dn] [,Vn]

opens a text file so that DOS can write to it or read from it. Unless you include a POSITION or APPEND command in your program, you must use the OPEN command before you use a text file. When fn does not exist, DOS creates a text file with that name. When fn exists, OPEN checks that the file is a text file.

CLOSE [fn]

tells DOS that you have finished reading from and writing to a file. Before ending, a program must close all the files it opened.

WRITE fn [,Bn]

tells DOS which file to write to. Use the WRITE command after you open the file. DOS writes to fn until another DOS command specifies a new fn.

READ fn [,Bn]

tells DOS which file to read from and where to start reading. Use the READ command after you open the file. DOS reads from fn until another DOS command specifies a new fn.

POSITION fn [,Rn]

sets the current position within the file. The POSITION command lets you skip a specified number of fields in the text file before you read or write more information.

APPEND fn [,Sn] [,Dn] [,Vn]

opens an existing text file and moves the current position to the end of the file so that your program can add data.

Reminder: Only CLOSE can be used in immediate execution (that is, from the keyboard). All of the other commands must be used in deferred execution (that is, issued from a BASIC program).

Using Random-Access Text Files

75 4.1 Random-Access Text Files
76 4.1.1 Record Length
76 4.1.2 Writing to a Record
77 4.1.3 Reading From a Record
77 4.2 A Demonstration: The RANDOM Program
79 4.3 A Sample Random-Access Program
79 4.3.1 Controlling the Program
79 4.3.2 Storing Records
80 4.3.3 Writing a Record
82 4.3.4 Reading a Record
84 4.4 Commands Used With Random-Access Text Files
84 4.4.1 The OPEN Command - Deferred Execution
85 4.4.2 The CLOSE Command - Immediate or Deferred Execution
86 4.4.3 The WRITE Command - Deferred Execution
87 4.4.4 The READ Command - Deferred Execution
88 4.5 Command Summary

Using Random-Access Text Files

This chapter discusses random-access text files: how to structure and use them. It also includes two demonstration programs. The chapter ends with a discussion of the random-access commands and their syntax.

Reminder: A sequential-access text file is a series of fields that can vary in length. A random-access text file is a series of records, all the same length and containing one or more fields.

4.1 Random-Access Text Files

A random-access text file is like a notebook with an unlimited number of pages (records). Each page holds the same number of characters, but the characters can be distributed on an unlimited number of lines (fields in a record). With a random-access text file (or a notebook), you go to a specific record (or page) before reading or writing fields (or lines) of text.

To use random-access commands, you need to

- specify a record length when you open the file.
- specify the number of the record that you are going to write or read.

It's a Good Idea: To use your random-access text files effectively, keep detailed information about their structure, format, record length, and field length. You can put this information into a comment at the beginning of each program, for example

```
10 REM R/A FILE; RECORD LENGTH 5; ONE FIELD PER
     RECORD
```

Better still, include this information in the file name, for example

```
R/A FILE-L5F1
```

4.1.1 Record Length

Record length is the allowable number of characters per record.

When you open a random-access text file, assign it a **record length**: the number of characters that each record holds. For example, to open a random-access text file named STOCK SYMBOLS, use this command

```
30 PRINT D$;"OPEN STOCK SYMBOLS,L12"
```

The L argument specifies that the length of each record in the file STOCK SYMBOLS is 12.

You can also specify record length with a hexadecimal number, for example

```
30 PRINT D$;"OPEN STOCK SYMBOLS,L$C"
```

Warning
Do not specify record length with a hexadecimal number if the length is contained in a variable that is less than 10.

By the Way: When you open a random-access text file, you don't need to specify the total number of records for a file. DOS does this for you.

4.1.2 Writing to a Record

To use the WRITE command with random-access text files, you specify the number of the record you are going to write to. If a record with that number does not yet exist, DOS creates it, reserving enough space on the disk for that entire record. For example, to enter data into record 10 of the STOCK SYMBOLS file, prepare to write to it by using the command

```
50 PRINT D$;"WRITE STOCK SYMBOLS,R10"
```

By the Way: You can also specify the record with a hexadecimal number, for example `50 PRINT D$;"WRITE STOCK SYMBOLS,R$A"`.

Then write a PRINT statement to put characters into record 10 of the STOCK SYMBOLS file. Even if you enter only one character, the record will use the full length declared in the OPEN statement.

Warning

Be careful not to print more characters than a record can hold. DOS will put any extra characters in the next record. Remember that the carriage return at the end of a field is a character too.

4.1.3 Reading From a Record

The first record in a file is record 0.

When you use the READ command with a random-access text file, specify the number of the record you want to read from. For example, prepare to retrieve the seventh record of the STOCK SYMBOLS file by typing

```
90 PRINT D$;"READ STOCK SYMBOLS,R6"
```

Then read it with an INPUT or GET statement. If the record you want to read doesn't exist, you'll see an `END OF DATA` message after your first GET or INPUT statement.

4.2 A Demonstration: The RANDOM Program

The RANDOM program on the SAMPLE PROGRAMS disk uses a random-access text file to demonstrate an inventory-control scheme. RANDOM is a very simple small program: it can handle at most nine parts. Any Apple II computer can handle thousands of parts in an inventory.

RANDOM automatically runs the random-access text file APPLE PROMS. You can list one or all items in the inventory. You can also change items, either one at a time or all at once. Here's how it works:

With the SAMPLE PROGRAMS disk in a disk drive, type

```
RUN RANDOM
```

You'll see these three options:

```
     APPLE PROMS
COMMAND        NUMBER
-------        ------
LIST             1
CHANGE           2
EXIT             3
CHOOSE NUMBER (1 - 3) 1
```

Remember: Press RETURN each time you complete a response.

Press 1 and you'll see this message:

```
PART NUMBER 1-9 (0=ALL) 0
```

Press 0 to get a list of all the parts in the inventory. You'll see

```
PART#  NAME            SIZE  IN STOCK
-----  ----            ----  --------
  1    PARALLEL PRINT  256   500
  2    COMMUNICATIONS  256   1250
  3    (NOT AVAILABLE) 256   0
  4    (NOT AVAILABLE) 256   0
  5    DISK BOOT       256   432
  6    STATE MACHINE   256   460
  7    SERIAL PRINTER1 256   878
  8    SERIAL PRINTER2 512   741
  9    CENTRONICS      256   1290

PRESS THE RETURN KEY TO CONTINUE
```

When you're ready to return to the list of options, press RETURN.

Try out the various program options. Option 1 allows you to list parts by part number, one at a time as well as all at once.

Option 2 allows you to change any or all part names and descriptions. For example, suppose part 3 should be COSMIC GLUE, size 56, with 1234 in stock. To revise the entry for part 3, select option 2 (CHANGE), then select part number 3. The old part name will be displayed, with the cursor at its beginning to allow you to enter the new name. After you type in the new name and press RETURN, the cursor will move to the right and perform similarly for part size and quantity. To use the current name, size, or quantity, just press RETURN.

Option 3 stops the program.

4.3 A Sample Random-Access Program

The ADDRESS program on the SAMPLE PROGRAMS disk illustrates the use of random-access text files. ADDRESS maintains a mailing list by doing two tasks: adding new addresses and looking up addresses that are already entered. Each task is written as a subroutine.

4.3.1 Controlling the Program

The main program is straightforward:

```
10 D$ = CHR$ (4)              (D$ is CONTROL-D)
20 F$ = "BLACK.BOOK"          (F$ is the file of addresses)
30 HOME                       (Clear the screen)
40 PRINT "WHAT WOULD YOU LIKE TO DO?"
50 PRINT " 1 ENTER A NEW ADDRESS"
60 PRINT " 2 LOOK UP AN ADDRESS"
70 PRINT " 3 END"
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 30
                              (Bad entry, try again)
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
```

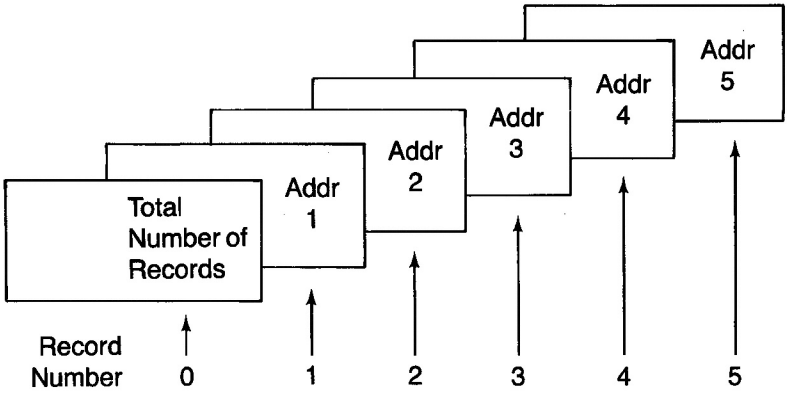
Lines 80 through 100 let you choose to enter a new address, read an existing address, or quit. If you type a letter to choose an option, the VAL statement (line 90) converts it to a number. Line 100 directs the program to repeat the question (line 40) if the number isn't in the expected range. Lines 110 through 130 send the program to the subroutine that corresponds to the option you select.

4.3.2 Storing Records

When you create a random-access text file, decide how information will be arranged within each record and how long a record will be. For example, ADDRESS stores information in a data file called BLACK.BOOK. Each record in BLACK.BOOK is 200 characters long and contains six fields: name, address, city, state, ZIP code, and telephone number.

ADDRESS reads each field into a separate variable (N$, A$, C$, S$, Z$, and P$). Then it adds 1 to the total number of records (TR). ADDRESS stores the first address in record 1, the second in record 2, and so on. In record 0 it stores the total number of records in the file. Figure 4-1 shows BLACK.BOOK's structure when it contains five addresses.

Figure 4-1. Five Addresses in BLACK.BOOK



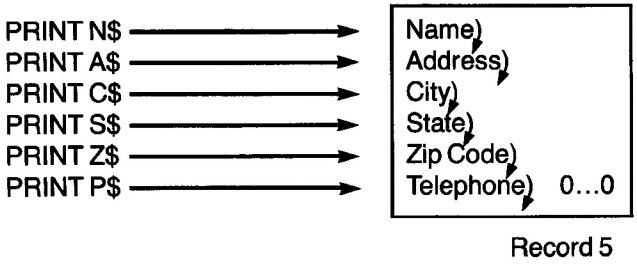
4.3.3 Writing a Record

To write a new record, ADDRESS first uses OPEN and WRITE to specify that BLACK.BOOK will receive the new record. Then a separate PRINT statement places each field in the record and prints the new total number to record 0.

Figure 4-2 shows the BASIC statements that place an address in record 5 of BLACK.BOOK. The) notation indicates a carriage return character and 0...0 represents the number of ASCII zeros needed to fill the record to 200 characters.

Figure 4-2. Writing an Address to Record 5

```
PRINT D$: "OPEN BLACK.BOOK,L200"
PRINT DS; "WRITE BLACK.BOOK,R5"
```



Lines 1010 through 1060 of the subroutine READ NEW INFO show the BASIC statements that gather the information for a record

```
1000 REM READ NEW INFO
1010 INPUT "NAME:      ";N$
                          (Read name to be entered)
1020 INPUT "ADDRESS:   ";A$
                          (Read address)
1030 INPUT "CITY:      ";C$
                          (Read city)
1040 INPUT "STATE:     ";S$
                          (Read state)
1050 INPUT "ZIP CODE:  ";Z$
                          (Read ZIP code)
1060 INPUT "PHONE:     ";P$
                          (Read phone number)
```

Lines 1070 through 1120 open the file whose name is stored in F$ and write information to the new record (record number TR):

```
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
1120 PRINT S$: PRINT Z$:    PRINT P$
                          in a separate field)
```

Lines 1130 through 1160 print to record 0 the total number of address records now in the file, close the file, and end the subroutine:

```
1130 PRINT D$;"WRITE ";F$;",R0"
                          (Prepare to write file into
                          record 0)
1140 PRINT TR             (Print new record number)
1150 PRINT D$;"CLOSE ";F$
                          (Close the file)
1160 RETURN               (End of subroutine)
```

4.3.4 Reading a Record

After you figure out how to store information in a record, decide how your random-access file will retrieve a particular record. The subroutine LOOKUP offers a solution: after the OPEN and READ commands specify BLACK.BOOK as the file to read, LOOKUP reads the total number of entries (record 0) in BLACK.BOOK. Then LOOKUP retrieves the names from the records and displays them in a numbered list. Finally, LOOKUP asks which record you want to display and displays the record you select.

Figure 4-3 shows how the BASIC statements relate to the fields in record 5 of the file BLACK.BOOK.

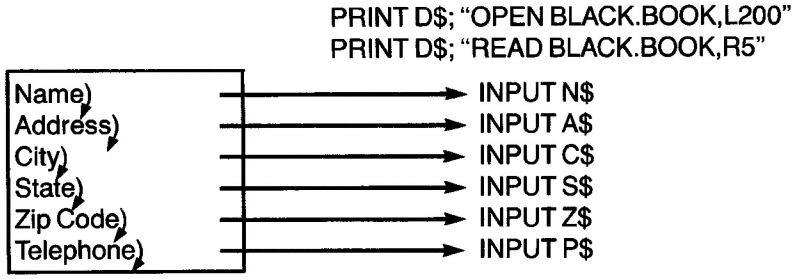


Figure 4-3. Reading an Address from Record 5

Lines 2000 to 2030 read the total number of addresses from record 0 of the file. Lines 2050 to 2100 read the name from each record and display the name on the screen.

```
2000 PRINT D$;"OPEN ";F$;",L200"
                              (Open the file)
2010 PRINT D$;"READ ";F$;",R0"
                              (Read from record 0)
2020 INPUT TR                 (Get the number of records)
2030 IF TR = 0 THEN GOTO 2210
                              (Check for no records)
2040 HOME                     (Clear the screen)
2050 PRINT "WHOSE ADDRESS DO YOU WANT?"
2060 FOR I = 1 TO TR          (For each record I,
2070  PRINT D$;"READ ";F$;",R";I
                              position to record I,
2080  INPUT N$                and read the stored name)
```

```
2090  PRINT I,N$              (Display the record number and
                               name on the screen)
2100 NEXT I                   (Repeat for all records)
2110 PRINT D$                 (Empty DOS command turns off
                               previous READ command)
```

The READ statement in line 2010 reads characters from BLACK.BOOK rather than from the keyboard. To read from the keyboard, line 2010 must be cancelled. The DOS command in line 2110, which prints only a CONTROL-D, does this.

Line 2120 asks for a number and reads it into the string R$. Line 2130 converts R$ to a number (R); if you pressed a letter key, the VAL function converts it to zero. The next line compares the number to the valid address numbers. An invalid response redisplays the question (line 140).

```
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 2120
                              (If bad number, try again)
```

The last part of the LOOKUP subroutine displays an address. The READ statement (line 2160) sets the position to the requested record. The INPUT statement in line 2170 reads the six fields of the record and lines 2180 through 2200 print the six fields on the screen. Line 2220 prevents the address from being erased before you can read it.

```
2150 HOME                     (Clear 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, address)
2190 PRINT C$, S$             (Put city, state on same line)
2200 PRINT Z$: PRINT P$       (Print ZIP code, phone)
2210 PRINT D$;"CLOSE ";F$
                              (Close the file)
2220 VTAB 23: HTAB 10         (Position cursor; save screen)
2230 INPUT "PRESS RETURN TO CONTINUE";T$
2240 RETURN
```

4.4 Commands Used With Random-Access Text Files

Except for CLOSE, the commands used with random-access text files work in deferred execution only.

Sequential-access text files, see Chapter 3.

Read Carefully: The commands described here have different options than those for sequential-access text files.

4.4.1 The OPEN Command - Deferred Execution

Before a program can write to or read from a random-access text file, it must open the text file with the OPEN command. The general form of the command is

OPEN fn, Ln [,Sn] [,Dn] [,Vn]

fn	is the name of the file to be opened for random access. When fn exists, DOS checks to see that it is a text file. When the file already exists and is open, DOS closes the named file and opens it again. If the file does not yet exist, DOS creates it as a text file and opens it.
Ln	where n is a number from 1 to 32767, specifies the number of bytes in each record of the file. You must specify the record length whenever you open the file; that is, whenever you create or reopen it. If you omit L, DOS uses 1.
[,Sn]	where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.
[,Dn]	where n is 1 or 2, specifies the drive containing the disk on which to store fn. If omitted, DOS uses the default drive number.
[,Vn]	where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS ignores the volume specification.

The first time you open a random-access text file, create it with a specified record length. Each time you subsequently open that file, you must specify that same length.

Warning

The declared record length must be greater than the maximum number of characters, *including carriage returns*, you're going to put in one of the file's records. If you write past the end of a record, you'll write over the beginning of the next record in the file, destroying what is there.

File buffer = 595 bytes

Whenever any text file is opened, DOS prepares to read or write starting at the beginning of the file and designates a file buffer to hold information about the file.

4.4.2 The CLOSE Command—Immediate or Deferred Execution

After writing to or reading from a file, a program must close it to ensure that DOS writes all the characters and releases the buffer associated with the file.

CLOSE takes the form:

CLOSE [fn]

where fn indicates the name of the file to close. When you omit the file name, the CLOSE command closes all open files, finishes any writing that your program is doing, and releases all file buffers.

Sometimes a program contains an error and stops before it can close all open files. Whenever this happens, issue the CLOSE command from the keyboard to close any files that are open.

A Programming Technique: Code into your program an ONERR GOTO statement that branches to a routine that will close any open files.

4.4.3 The WRITE Command—Deferred Execution

The WRITE command names the random-access file you're writing to, identifies the record that receives information, and specifies the position within the record of the first character to be written.

The WRITE command takes this form:

WRITE fn [,Rn] [,Bn]

fn	indicates the name of the file to be written to.
[,Rn]	where n is a number from 0 to 32767, indicates the record at which the program is going to start writing. If Rn is omitted, DOS prepares to start writing at record 0.
[,Bn]	where n is the number of characters that DOS should skip in the specified record, changes the current position in the file. If this option is omitted, DOS prepares to start writing at byte 0, the beginning of the record specified by R.

WRITE remains in effect until the next DOS command or the next INPUT or GET statement.

Use WRITE before putting characters into a record with PRINT statements, and use WRITE each time you want to print to a record other than the current one.

Warning
If you are not using ONERR GOTO and your program generates an error message, the message is stored as the last field in your text file. Also, the error message cancels the WRITE command.

The WRITE statement tells DOS to send to the open file all the characters that are normally sent to the screen. This includes error messages—if you have not included an ONERR GOTO statement—and the prompt characters of INPUT statements—if INPUT appears after the WRITE command.

Warning

Do not use the sequential-access command POSITION to skip fields within a record. If you do, you cancel the WRITE command.

Do not print more characters (including the carriage-return character) to a random-access record than you specify in the L argument of OPEN. DOS will write the current record correctly but will calculate the next record's starting position as if the previous record had been within the specified length. The next record will write over the last characters of the previous, oversized record, including the carriage-return character marking the end of that record.

If you print fewer characters to a random-access record than you specify in the L argument of OPEN, you create two new fields: the field you just printed, followed by the end of the field you were writing over.

4.4.4 The READ Command—Deferred Execution

When used with random-access files, the READ command identifies the file from which the next INPUT or GET statement takes characters. You can specify the record (Rn) within the file and the byte position (Bn) within the record from which the first character will be read.

The READ command remains in effect until the next DOS command and takes this form:

READ fn [,Rn] [,Bn]

fn	indicates the name of the file to be read.
[,Rn]	where n is a number from 0 to 32767, indicates the record from which the program is going to read. If Rn is omitted, DOS prepares to start reading at record 0. If Rn indicates a record that doesn't exist and you are not using ONERR GOTO, you'll see an END OF DATA message after the first INPUT or GET statement from the nonexistent record.
[,Bn]	where n is the number of characters that DOS is to skip in the specified record, changes the position in the file relative to the file's current position. If this option is omitted, DOS prepares to start reading from byte 0, the beginning of the record specified by R.

Caution: Unless D$ = CHR$(13) + CHR$(4), avoid using the Applesoft command TRACE with INPUT statements—your program will stop and wait for input from the keyboard.

4.5 Command Summary

OPEN fn, Ln [,Sn] [,Dn] [,Vn]

opens a random-access text file so that DOS can write to it. The length (L), specifies the number of bytes, or characters, each record can hold. If the file is a new one—that is, if it doesn't yet exist—DOS creates it and gives it the file name you specify in fn. If fn does exist, DOS checks that it is a text file.

CLOSE [fn]

tells DOS that you have finished reading from and writing to a file. If fn is omitted, DOS closes all open files.

WRITE fn [,Rn] [,Bn]

specifies the random-access file to write to and where to start writing: the number of the record (Rn) and the position within the record of the first character to write (Bn). The file must be open. DOS writes to fn until the next DOS command or the next INPUT or GET statement.

READ fn [,Rn] [,Bn]

specifies the random-access file to read from and where to start reading: the number of the record (Rn) and the position within the record of the first character to read (Bn). The file must be open. DOS reads fn until the next DOS command.

Reminder: All of these commands, including CLOSE, can be issued from a BASIC program. Only CLOSE can be issued from the keyboard.

Programming With More Sophistication

91 5.1 Using a Command File
92 5.1.1 The EXEC Demonstration
94 5.1.2 Creating a Command File
96 5.1.3 Special-Purpose Command Files
96 5.1.3.1 Capturing Lines From a BASIC Program
98 5.1.3.2 Translating Machine Language to BASIC
99 5.2 The EXEC Command
101 5.3 The MAXFILES Command
101 5.3.1 Buffering Information
102 5.3.2 Example
103 5.4 CHAIN
103 5.4.1 The CHAIN Command - Integer BASIC
105 5.4.2 The CHAIN Program - Applesoft
106 5.5 The MASTER Program
107 5.5.1 Example
108 5.6 Making a Turnkey Disk
108 5.6.1 Creating a Turnkey Disk
109 5.6.2 Creating a Master Turnkey Disk
109 5.6.3 A Disk for All Systems
110 5.6.3.1 Example
113 5.7 Summary
113 5.7.1 Commands
113 5.7.2 Program

Programming With More Sophistication

This chapter describes three commands and two programs that allow DOS to interact with several programs, files, and systems.

- EXEC lets you use a command file to execute another file.
- CHAIN lets you link one program to another.
- MAXFILES lets you increase or decrease the number of files that can be open in your Apple II's memory.
- The MASTER program lets you create a master disk that can operate in Apple II computers with different memory sizes.

5.1 Using a Command File

Command file = EXEC file.
Command file has file type T.

Your Apple II computer can take its commands from a special text file, called a **command file**, as well as from the keyboard. Command files let you automate a frequently used set of commands, convert a program written in one BASIC dialect to the other, or repeatedly enter the same information into a program. To use this programming capability, first write a BASIC program that creates a command file, then execute the command file with the EXEC command.

A command file may contain anything that you might type in from the keyboard: DOS commands, lines of BASIC statements, or even Monitor commands. When you put a DOS command in a command file, you do *not* precede it with CONTROL-D.

For more on EDASM, see the *Applesoft/ DOS Tool Kit*.

You can't look at a command file with a LIST statement. Instead, use the program GET TEXT, on the SAMPLE PROGRAMS disk, or a special editing program, like Apple Writer 2.0 or EDASM. You can also issue a MON C command to watch as the command file issues and executes its commands.

The following demonstration uses GET TEXT to look at the command file DO'ER.

Remember: Do not precede a DOS command in a command file with CONTROL-D.

5.1.1 The EXEC Demonstration

This demonstration has two parts. First run the BASIC program EXEC DEMO on the SAMPLE PROGRAMS disk to create a command file. Then issue the EXEC command so your computer takes its instructions from the command file instead of the keyboard.

By the Way: Be sure your copy of the SAMPLE PROGRAMS disk is write-enabled so you can save the command file on it.

1. To create the command file, insert your SAMPLE PROGRAMS disk into drive 1 and type

   ```
   RUN EXEC DEMO, D1
   ```

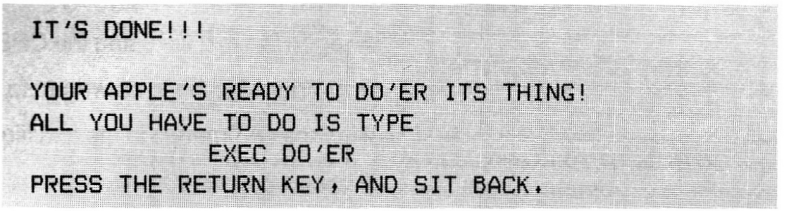
   Read the full page of instructions (Figure 5-1).

Figure 5-1. The First Screen of the EXEC Demonstration

```
            << EXEC DEMO >>

THIS PROGRAM CREATES A SEQUENTIAL TEXT
FILE NAMED "DO'ER" CONTAINING SEVERAL
STRINGS, EACH A LEGAL APPLE II COMMAND.

THEN YOU TYPE:
EXEC DO'ER
THEN THE COMMANDS IN FILE DO'ER TAKE
CONTROL OF YOUR COMPUTER. EACH COMMAND
WILL BE EXECUTED JUST AS IF IT HAD BEEN
TYPED AT THE KEYBOARD. THE DOS MANUAL
DESCRIBES THE PROGRAM IN MORE DETAIL.

        << HAPPY EXECUTING >>

PRESS THE SPACE BAR TO MAKE THIS
PROGRAM CREATE THE FILE DO'ER.

IF YOU WISH TO STOP THIS PROGRAM NOW,
YOU MAY PRESS THE ESC KEY.
```

2. After you've read EXEC DEMO's first screen, press the SPACE bar. EXEC DEMO writes the DO'ER file to the disk and then displays the screen shown in Figure 5-2.

Figure 5-2. EXEC DEMO's Final Screen



EXEC DEMO has created a command file, named it DO'ER, and put DO'ER on the disk.

3. To see a command file in action, execute DO'ER by typing

   EXEC DO'ER

   DOS loads the first sector of DO'ER from the disk and begins executing the commands. DO'ER describes everything it is doing. Since DO'ER runs some programs that use Integer BASIC, you'll see LANGUAGE NOT AVAILABLE if your computer doesn't have that language. Nothing to worry about; DO'ER executes those programs it can. And you won't have to touch the keyboard unless your catalog has more than eighteen entries. Then you'll have to press RETURN to see the nineteenth.

GET TEXT reads a text file; see Chapter 3 for a complete description.

4. To look at the commands in the DO'ER file, use the GET TEXT program. Type

   RUN GET TEXT

   When GET TEXT asks LIST WHAT TEXT FILE?, type

   DO'ER

   As GET TEXT displays the contents of DO'ER, notice the wide variety of commands it contains.

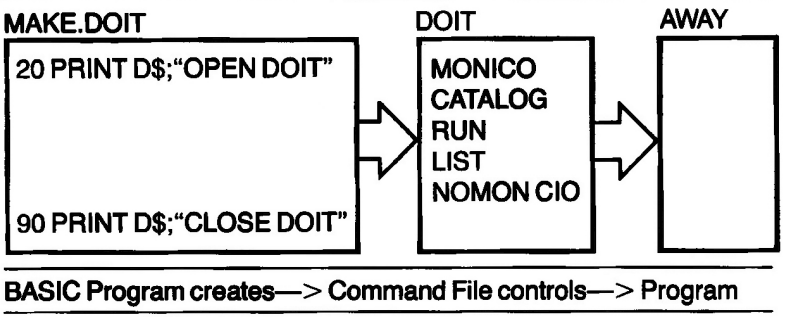
5.1.2 Creating a Command File

To create a command file, a BASIC program must:

- open the text file using the OPEN command,
- prepare the file to be written to using the WRITE command,
- place commands in the text file using PRINT or LIST,
- close the text file.

Here's a step-by-step example that illustrates how to create an EXEC file. Figure 5-3 shows the files and commands the example uses. The files in the example are MAKE.DOIT, a BASIC program that creates a command file, and DOIT, a command file that executes the program called AWAY. DOIT will contain the following DOS commands: CATALOG, RUN, and LIST.

Figure 5-3. A Command File



1. First, type in this program:

```
5  REM MAKE.DOIT
10  D$ = CHR$ (4)       (REM D$ is CONTROL-D)
20  PRINT D$;"OPEN DOIT"
                        (Prepare the file to be written to)
30  PRINT D$;"WRITE DOIT"
40  PRINT "MONICO"      (Watch the action)
50  PRINT "CATALOG"     (Put these three commands
60  PRINT "RUN AWAY"    into the command
70  PRINT "LIST"        file DOIT)
80  PRINT "NOMON CIO"   (Turn off MON)
90  PRINT D$;"CLOSE DOIT"
                        (And close the file)
```

   Save the program on the disk and name it MAKE.DOIT by typing

```
SAVE MAKE.DOIT
```

   The PRINT statements beginning with D$ (lines 20, 30, and 90) are DOS commands that will be executed when you run MAKE.DOIT. The other PRINT statements will be written to the command file DOIT to be used later. Notice the DOS commands that will be put in the command file: none are preceded by CONTROL-D.

2. Next, type in this Applesoft 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."
```

   Save the program on the disk by typing

```
SAVE AWAY
```

   Later, the command file will run AWAY.

3. After you have saved MAKE.DOIT and AWAY, run MAKE.DOIT and create the sequential-access text file named DOIT by typing

   ```
   RUN MAKE.DOIT
   ```

   When MAKE.DOIT has finished, DOIT is on the disk.

4. To watch those commands race by as if you were typing them in very quickly from the keyboard, and to execute the commands in the file DOIT one by one, type the commands

   ```
   MON I,C,O
   EXEC DOIT
   ```

   The command file DOIT displays the files on the disk, the sentences printed out by the program, and a listing of the program AWAY.

MAKE TEXT, see Chapter 3.

Note: You can also create a command file by using the program MAKE TEXT. However, you have no way to fix an erroneous line in MAKE TEXT.

5.1.3 Special-Purpose Command Files

You can design command files to do many tasks: convert a listing of a BASIC program into a text file, edit a program using a DOS-compatible word processor, place part of a program anywhere in another program, insert subroutines from a subroutine file into a program, even connect two programs. The following examples illustrate two of these tasks.

5.1.3.1 Capturing Lines From a BASIC Program

The EXEC command lets you capture lines from a program in memory and insert a copy of them into a text file to create a new program. The "captured" lines may need some editing before the new program can run, but editing existing lines may be easier than a lot of retyping.

The next example shows you the CAPTURE routine, which reads lines of a BASIC program in memory and puts them into a text file named LISTING.

```
1 REM CAPTURE ROUTINE
2 D$ = CHR$ (4) : POKE 33,33
                              (REM D$ is CONTROL-D
                              Change the text area to prevent
                              spaces in PRINT statements)
3 PRINT D$; "OPEN LISTING"
                              (Prepare the file LISTING to
                              write to)
4 PRINT D$; "WRITE LISTING"
5 LIST 2270,5130              (List the lines to the file)
6 PRINT D$; "CLOSE LISTING"
                              (Close up the file,
7 TEXT : END                  and reset the text area)
```

The POKE statement in line 2 puts a value into a memory location. POKE 33,33 sets the right margin of the text area 33 characters away from the left margin so that BASIC won't split a long line into two and DOS won't add extra spaces.

Use the command EXEC LISTING to read the lines of LISTING into the computer's memory as if you had typed them in.

You can tailor these lines to your own purposes by adding the CAPTURE routine to the beginning of your own program. Replace the line numbers in line 5 with the numbers of the lines you want to capture from the program in memory; replace LISTING (lines 3, 4, and 6) with the name of the new file to which you want to add the captured lines.

You can also use the CAPTURE routine to combine programs. First, create a program using CAPTURE. Then load the second program into memory and use the EXEC command on the file that was created using CAPTURE.

> **Remember**: Using the EXEC command on a file does not delete the program that is already in memory.

5.1.3.2 Translating Machine Language to BASIC

PEEK returns a value stored in a memory location. POKE stores a value directly into memory.

The demonstration program POKER, on the SAMPLE PROGRAMS disk, creates an EXEC file that reads a binary program and translates it into BASIC. POKER uses PEEK to read consecutive bytes of the binary program, and for each byte, it prints a POKE statement into a command file.

When you execute the command file, it builds a new BASIC program containing these POKE statements. You can also write the program to have the command file add BASIC statements to an existing BASIC program in memory.

```
100 REM PROGRAM POKER
110 D$ = CHR$ (4)          (D$ is CONTROL-D)
120 HOME
130 INPUT "FILE NAME TO CREATE? ";N$
140 PRINT : INPUT "STARTING ADDRESS (DECIMAL)?
    ";S
150 PRINT : INPUT "ENDING ADDRESS (DECIMAL)?
    ";E
160 IF S>E THEN PRINT : PRINT "THE STARTING
    ADDRESS MUST BE LESS THAN" : PRINT "THE
    ENDING ADDRESS"; CHR$(7): GOTO 140
170 PRINT : INPUT "BASIC LINE  TO START AT?
    ";LINE
180 IF LINE<0 OR LINE>63999 THEN PRINT : PRINT
    "BASIC LINE NUMBERS MUST BE IN THE
    RANGE": PRINT "OF 0 TO 63999"; CHR$(7):
    GOTO 170
190 PRINT D$;"OPEN "N$     (Opens, closes, and deletes
200 PRINT D$;"CLOSE "      the file you named in 130
210 PRINT D$;"DELETE "N$   in case it already exists)
220 PRINT D$;"OPEN "N$     (Opens file named in 130)
230 PRINT D$;"WRITE "N$    (Prepares to write to it)
240 FOR PLACE = S TO E     (For each memory location,
250 C = C + 1              increments counter, puts
260 IF C = 10 THEN C = 1   ten POKEs on each line)
270 IF C < > 1 THEN 300    (For first POKE on a line,
280 PRINT : PRINT LINE;    prints the line number, and
290 LINE = LINE + 1        increments it)
300 PRINT "POKE ";PLACE;" , "; PEEK
    (PLACE);" :";          (Pokes a byte
310 NEXT PLACE             and handles next location)
320 PRINT                  (Creates a new line for DOS
                           command)
330 PRINT D$;"CLOSE"       (Then closes the file)
340 END
```

To use this program, put the SAMPLE PROGRAMS disk in your disk drive and type

```
RUN POKER
```

When POKER asks, enter the name you'd like to use for your command file (line 130), enter the memory locations (lines 140 and 150), and enter the BASIC line number (line 170). If you name the command file FILLER, you can execute it by typing

```
EXEC FILLER
```

FILLER will add the lines containing the POKE statements to whatever BASIC lines are already in memory.

5.2 The EXEC Command

The EXEC command directs DOS to take its commands from a command file rather than from the keyboard. You can start execution at any line of a command file.

The EXEC command has the form

EXEC fn [,Rn] [,Sn] [,Dn] [,Vn]

fn — indicates the name of the command file, a text file that can contain DOS commands, data, and BASIC statements.

[,Rn] — where n is a number from 0 to 32767, indicates the number of lines to skip in the command file. When Rn is omitted, DOS starts executing the command file at the first line. DOS counts fields from the beginning of the command file, so Rn is always relative to 0. Specifying a value for R beyond the end of the file returns an END OF DATA message.

[,Sn] where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.

[,Dn] where n is 1 or 2, specifies the drive that holds the disk containing the command file. If omitted, DOS uses the default drive number.

[,Vn] where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS ignores the volume specification. You can execute only one command file at a time.

Warning

Be careful if your command file is on a disk in one drive and the disk containing the program the command file will run is in another drive. The drive containing the EXEC file should be the default so that DOS can return to the EXEC file after each sector of the program is executed.

When a command file completes all its commands, it closes itself and stops. If a command file issues the EXEC command to call another command file, the original command file closes, and the new command file opens and executes. The second command file replaces the first in memory.

The command file currently in memory is not affected by the NEW statement or by the FP, INT, or CLOSE commands.

Using the EXEC command on a command file does not delete a program that is already in memory.

You can stop a command file that is executing BASIC statements by pressing CONTROL-C; the remaining commands in the command file will not be executed. To stop a command file that is executing DOS commands, turn off the computer.

Warning

When a command file is running a program, an INPUT statement in the program will read the next field from the command file rather than from the keyboard. If the input is an immediate-execution DOS command, the command is executed before the program continues.

5.3 The MAXFILES Command

At startup DOS gives you a maximum of three files that can be open, or active, at one time. But you may want more if you are building a large program. Or you may want less if you need every possible byte in memory.

The MAXFILES command allows you to increase the number of active files to a maximum of 16 or decrease them to a minimum of one. The number you specify in the command is the maximum until you issue another MAXFILES command or start up DOS again.

The syntax for the MAXFILES command is

MAXFILES n

where n is an integer from 1 to 16, specifies the number of file buffers that can be active at one time. If you specify a value beyond this range, you'll see a RANGE ERROR message.

5.3.1 Buffering Information

For each file you open, DOS sets aside 595 bytes of memory space in an area called a **file buffer**. DOS uses 256 bytes of the file buffer for data and 256 bytes for the track/sector list. The remaining 83 bytes are for "housekeeping" information such as the file's location on the disk.

Track/sector list: see Appendix B.

When your program reads information from a disk, DOS brings in 256 bytes, putting them in the data section of the buffer. DOS delivers to your program whatever subset of those 256 bytes your program requests.

1 byte = 1 character

When your program sends information to a disk, DOS first stores characters in the data section of the buffer until it accumulates 256 bytes. Then DOS writes them to the disk all at once.

By the Way: DOS initializes the data section by filling it with 256 zeros; these are replaced by the characters you enter. When DOS writes that buffer to a disk, it transfers the entire 256 bytes. If you've entered fewer than 256 characters, DOS transfers the characters you entered and any zeros remaining in the buffer.

File buffer = 595 bytes in memory

The three file buffers that DOS initially supplies reserve a total of 1785 bytes in memory. When you execute MAXFILES to increase the number of files, DOS reserves 595 additional bytes of memory for each new file buffer. When MAXFILES decreases the number of active files, DOS releases 595 bytes of memory for each buffer no longer required.

You can issue the MAXFILES command in immediate execution before loading and running a program.

Warning

When you issue the MAXFILES command from the keyboard, DOS moves HIMEM in memory, but does not move any Integer BASIC program or Applesoft strings.

Changing the number of buffers erases Integer BASIC programs. Increasing the number of buffers affects the area where Applesoft strings are stored.

HIMEM = highest memory address available to an Applesoft program; see Chapter 6.

By the Way: You can also issue MAXFILES from an Applesoft program. Make MAXFILES the first program statement since it changes memory pointers for string variables.

All DOS commands except PR#, IN#, and MAXFILES require a file buffer. If you set MAXFILES to 1 and open a file, you use the present limit, one buffer. You'll see `NO BUFFERS AVAILABLE` if you then issue a DOS command other than PR#, IN#, or MAXFILES.

5.3.2 Example

To change MAXFILES from an Applesoft program, make the MAXFILES command the first statement in the program, before you declare any string variables. For example

```
10 PRINT CHR$(4); "MAXFILES 5"
```

5.4 CHAIN

CHAIN lets you link one program with another. When both programs are in Integer BASIC, use the CHAIN command. When both programs are in Applesoft, use the CHAIN program on the SYSTEM MASTER disk.

5.4.1 The CHAIN Command—Integer BASIC

When an Integer BASIC program is too big to fit entirely in memory, the CHAIN command lets you bring in parts of the program and run each part, one at a time. When the second part comes in, DOS does not clear the variables or close the files that were used by the first part.

The command takes this form

CHAIN fn [,Sn] [,Dn] [,Vn]

fn	indicates the file containing the Integer BASIC module to be run next.
[,Sn]	where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.
[,Dn]	where n is 1 or 2, specifies the drive containing the disk on which to store fn. If omitted, DOS uses the default drive number.
[,Vn]	where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS uses the default volume number.

When you chain from one program to another, the first program is removed from memory. Execution begins at the lowest numbered line of the next program. To use the first program again, use the CHAIN command again.

Warning

If there is an array in two chained programs, be sure the dimensions are declared in the same program that contains the array.

If two chained programs use a defined function, be sure it is defined in each.

In the following example, the program PART1 uses the CHAIN command to connect a second program part, PART2.

Reminder: Be sure you are running Integer BASIC. If you don't see a > prompt character, issue the INT command.

1. Type in this program. Type a CONTROL-D character after the first quotation mark in line 50.

```
10 REM PART1
12 DIM I$(30)              (In Integer BASIC, a string
                           variable must be dimensioned if
                           it is longer than one character.)
30 I$="THE STRING I$ IS PRESERVED."
                           (Set a string value)
40 PRINT "PART1: I$ HAS BEEN SET."
                           (And say so)
50 PRINT "CHAIN PART2 "
                           (Chain to PART2)
60 END
```

   To save this program, type

```
SAVE PART1
```

2. Now type in and save PART2.

```
15 REM PART2
45 PRINT "PART2: ";I$      (Prints the string set in PART1)
60 END
```

3. If you want to see what will be printed, type

```
RUN PART1
```

   The CHAIN command works properly if the variable I$ retains the value set by PART1 when it is printed out by PART2. Then you'll see `PART2: THE STRING I$ IS PRESERVED.` If CHAIN doesn't work, you'll see only `PART2:`.

5.4.2 The CHAIN Program—Applesoft

This example uses the binary CHAIN program on the SYSTEM MASTER disk to show you chaining in Applesoft. PART1 sets the variable I$ and issues the CHAIN command to link with PART2.

Reminder: Be sure the disk you're using to hold PART1 and PART2 also has the CHAIN program. If it doesn't, use the FILEM program to copy it from the SYSTEM MASTER disk.

1. Type in PART1 and save it.

Warning

When you type line 60, do not type a space, a comma, a colon, or a semicolon after the number 520, the memory address where the CHAIN subroutine was loaded.

```
10 REM PART1
20 D$=CHR$(4)              (D$ is set to CONTROL-D)
30 I$="THE STRING I$ IS PRESERVED."
                           (Set a string value and announce
                           it)
40 PRINT "PART1: I$ HAS BEEN SET."
50 PRINT D$;"BLOAD CHAIN, A520"
                           (Load to memory address)
60 CALL 520"PART2"         (Chain to PART2)
```

2. Now type in and save PART2.

```
15 REM PART2
45 PRINT "PART2: ";I$
60 END
```

The CHAIN command in line 50 of PART1 allows PART2 to display the contents of I$ (line 45).

3. To put the CHAIN program through its paces, type

```
RUN PART1
```

The CHAIN command works properly if the variable I$ retains the value set by PART1 when it is printed out by PART2. Then you'll see `PART2: THE STRING I$ IS PRESERVED.` If CHAIN doesn't work, you'll see only `PART2:`.

5.5 The MASTER Program

You can skip this section if you aren't going to use your disks on other computers with a smaller memory.

You place a copy of DOS on every disk you initialize. When you start up the disk, DOS goes into the same memory locations it had in the system on which you initialized the disk. If you start up the disk on a computer of a different size, DOS won't work. So, if you use more than one Apple II computer and they have different sizes of memory, a master disk is handy.

A master disk contains a relocatable image of DOS and can be used on any Apple II computer with at least 16K of memory. When you start up a master disk, DOS is placed in locations relative to the size of the system on which you are using the disk.

MASTER is an Applesoft program that executes MASTER CREATE, a binary program on the SYSTEM MASTER disk that converts an initialized, write-enabled disk into a master disk. Converting the disk will not harm any programs already on it.

MASTER requests the name of a program that DOS will run each time the disk is started. You may name a traditional greeting program or you may name some other program.

By the Way: Earlier manuals said to issue a binary command to get MASTER CREATE running. You can still execute it directly by typing

```
BRUN MASTER CREATE
```

Warning

After using the MASTER program, always restart DOS before doing any other work.

5.5.1 Example

1. Write a greeting program that reflects the master status the disk will have, for example:

```
10 PRINT "MASTER DISK 32K"
20 END
```

2. Save the greeting program on the disk that you wish to convert. Type

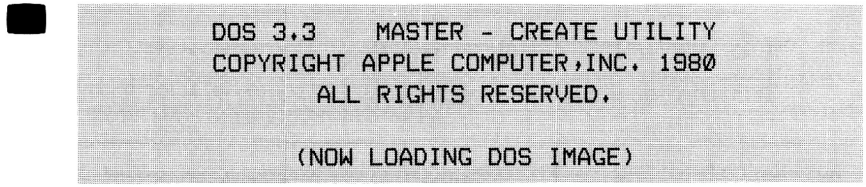
```
SAVE HELLO
```

When the IN USE light goes out, remove that disk from the drive.

3. Insert the SYSTEM MASTER disk in drive 1 and type

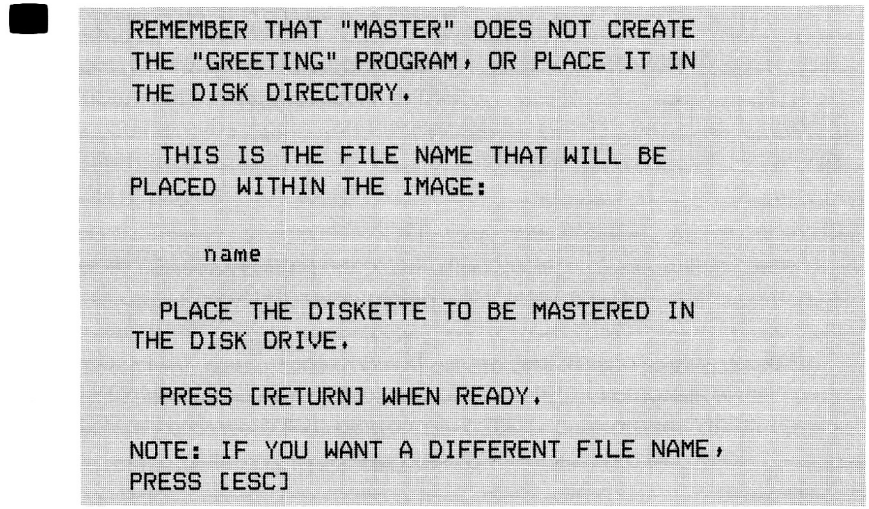
```
RUN MASTER
```

The message EXECUTING MASTER CREATE appears briefly. Then you'll see the title screen of the MASTER CREATE program:



```
DOS 3.3    MASTER - CREATE UTILITY
COPYRIGHT APPLE COMPUTER,INC. 1980
        ALL RIGHTS RESERVED.

      (NOW LOADING DOS IMAGE)
```

4. In response to the message, PLEASE INPUT THE "GREETING" PROGRAM'S FILE NAME:, enter the name of your greeting program. You'll see this screen:



```
REMEMBER THAT "MASTER" DOES NOT CREATE
THE "GREETING" PROGRAM, OR PLACE IT IN
THE DISK DIRECTORY.

  THIS IS THE FILE NAME THAT WILL BE
PLACED WITHIN THE IMAGE:

     name

  PLACE THE DISKETTE TO BE MASTERED IN
THE DISK DRIVE.

  PRESS [RETURN] WHEN READY.

NOTE: IF YOU WANT A DIFFERENT FILE NAME,
PRESS [ESC]
```

If you'd like the disk to run a program other than "name" each time the disk is started, press (ESC) and enter the name of your greeting program.

5. Remove the SYSTEM MASTER disk from drive 1 and insert the initialized disk that you wish to convert to a master disk. Press (RETURN). MASTER begins the conversion and informs you when the process is complete.
6. When the conversion is done and the IN USE light on the disk drive goes out, press (ESC), remove the disk, and label it. Indicate that it's a master disk.

Reminder: Don't forget. After using the MASTER program, restart DOS.

5.6 Making a Turnkey Disk

A **turnkey disk** automatically runs a program when it starts up DOS. This section tells you how to make three kinds of turnkey disks: a simple startup disk, a master startup disk, and a master startup disk that brings the alternate BASIC into all Apple II systems, whatever the memory size.

Warning
A turnkey disk does not automatically return from DOS to BASIC unless its turnkey program provides an exit command.

To make a turnkey disk, load your turnkey program into memory and save it on an initialized disk. When you issue the SAVE command, use the name of the disk's greeting program.

5.6.1 Creating a Turnkey Disk

This example shows you how to create a turnkey disk so that the RENUMBER program runs automatically every time you start that disk.

1. Initialize a blank disk, using the name HELLO for the greeting program.
2. Insert the SYSTEM MASTER into drive 1 and bring RENUMBER into memory by typing

   ```
   LOAD RENUMBER
   ```

3. Remove the SYSTEM MASTER disk and insert the initialized disk into your disk drive. Put the RENUMBER program on it by typing

   ```
   SAVE HELLO
   ```

Now the RENUMBER program will be named HELLO. Any time you start up with this disk, you'll automatically start HELLO, the program that used to be called RENUMBER.

5.6.2 Creating a Master Turnkey Disk

MASTER program, see section 5.5.1.

Use the MASTER program to turn an initialized disk into a master disk. When MASTER asks you to enter the name of the greeting program, enter the name of the program that you want to be the turnkey program.

5.6.3 A Disk for All Systems

Suppose you want to make a turnkey disk that will run on all Apple II computers. You'll need a DOS that works in any memory size and that brings in whichever BASIC your programs require. Look first at how each type of Apple II computer interacts with memory and with BASIC.

When you insert a DOS disk in drive 1 and turn on the power, your computer runs the greeting program on that disk, if it has in ROM the BASIC in which the greeting program is written. That is,

- the standard Apple II runs an Integer BASIC greeting program.
- the Apple II Plus and the Apple IIe run an Applesoft greeting program.
- the standard Apple II with the Applesoft Card and the Apple II Plus with the Integer BASIC Card can each run a greeting program written in either BASIC.

When the greeting program is written in the alternate BASIC and you don't have that BASIC in memory, you'll see `LANGUAGE NOT AVAILABLE` and the prompt character for the resident BASIC.

Use the SYSTEM MASTER disk to load the alternate BASIC.

If you have the Apple Language System, you can run programs in the alternate BASIC after loading the alternate BASIC into the Language Card.

> **By the Way:** Installing a language card increases your Apple II's memory by adding 16K bytes of memory. Your computer must have at least 48K before you can install a language card.

The simplest way to load the alternate BASIC is to start up DOS by using the SYSTEM MASTER disk. The HELLO program on the SYSTEM MASTER uses LOADER.OBJ0 to determine which BASIC is resident. Then it loads the alternate BASIC from the disk.

5.6.3.1 Example

This example shows you how to make a disk that starts on all the Apple II systems.

> **Remember:** If you are using more than one drive, you'll need to specify the drive option, [,Dn].

1. Load the HELLO program from the SYSTEM MASTER disk into memory. Use that greeting program when you initialize a blank disk with the INIT command. When INIT is finished, reinsert the SYSTEM MASTER disk.

2. Use FILEM to copy these language files and programs that load the languages from the SYSTEM MASTER to your new disk:

   APPLESOFT
   INTBASIC
   FPBASIC
   LOADER.OBJ0

   When you start this disk, DOS will load the alternate BASIC.

   Put any other programs you want on this disk.

3. Convert the initialized disk to a master disk with the MASTER program. Type

   ```
   RUN MASTER
   ```

   Insert the newly initialized disk in the drive when MASTER instructs you.

   When MASTER asks for the name of the greeting program, type

   ```
   HELLO
   ```

4. Change the HELLO program to run your turnkey program, called TURNKEY in this example.

   To change HELLO, unlock it, load it, and list it:

```
10 TEXT : HOME
20 D$ = CHR$ (4)   REM CTRL-D
30 VTAB 2:A$ = "APPLE II": GOSUB 1000
40 VTAB 4:A$ = "DOS VERSION 3.3 SYSTEM
     MASTER": GOSUB 1000
50 VTAB 7:A$ = "JANUARY 1, 1983": GOSUB 1000
60 PRINT D$;"BLOAD LOADER.OBJ0"
70 CALL 4096
80 VTAB 10: CALL - 958:A$ = "COPYRIGHT APPLE
     COMPUTER,INC. 1980,1982 ": GOSUB 1000
```

```
90 C = PEEK ( - 1101): IF C=6 THEN PRINT :
     INVERSE :A$ = "BE SURE CAPS LOCK IS
     DOWN": GOSUB 1000: NORMAL
100 PRINT CHR$(4);"FP"
1000 REM CENTER STRING A$
1010 B = INT (20 - ( LEN (A$) / 2)): IF B =<0
       THEN B=1
1020 HTAB B: PRINT A$: RETURN
```

5. Change line 100 so that it reads

```
100 PRINT CHR$(4);"RUN TURNKEY"
```

Then save the file with the name HELLO and lock it.

6. Unlock the file APPLESOFT, load it, and list it:

```
10 TEXT : CALL -936: DIM A$(40)
20 D$="":                          REM CTRL-D
30 VTAB 2:A$="APPLE II": GOSUB 1000
40 VTAB 4:A$="DOS VERSION 3.3 SAMPLE
     PROGRAMS": GOSUB 1000
50 VTAB 7:A$="JANUARY 1, 1983": GOSUB 1000
60 PRINT D$;"BLOAD LOADER.OBJ0"
70 CALL 4096
80 VTAB 10: CALL -958:A$="COPYRIGHT APPLE
     COMPUTER, INC. 1980, 1982": GOSUB 1000
90 C=PEEK (-1101): IF C<>6 THEN GOTO 100:
     PRINT
95 POKE 50,127:A$ ="BE SURE CAPS LOCK IS
     DOWN": GOSUB 1000: POKE 50,255
100 PRINT D$;"INT"
1000 REM CENTER STRING A$
1010 B = 20 - ( LEN (A$) / 2): IF B = 0 THEN
       B=1
1020 TAB B: PRINT A$: RETURN
```

Change line 100 so that it reads

```
100 PRINT D$;"RUN TURNKEY"
```

Save the file with the name APPLESOFT and lock it. This changes HELLO and APPLESOFT so they run your turnkey program instead of the BASIC that they ran originally.

7. Rename your turnkey program TURNKEY. Here is a sample TURNKEY program in Applesoft:

```
100 TEXT : HOME
200 VTAB 3: PRINT "THIS IS MY TURNKEY
      PROGRAM "
300 VTAB 5: FOR A = 1 TO 10: PRINT A: NEXT A
400 PRINT : PRINT "THIS IS THE END OF MY
      TURNKEY PROGRAM"
500 END
```

8. After you make sure your disk has the HELLO, APPLESOFT, INTBASIC, FPBASIC, LOADER.OBJ0, and TURNKEY files, test your turnkey disk by putting it in drive 1 and turning the power off and on. The disk should start DOS, load Integer BASIC or Applesoft if necessary, and run the program named TURNKEY.

5.7 Summary

5.7.1 Commands

EXEC fn [,Rn] [,Sn] [,Dn] [,Vn]

directs DOS to take its commands from a command file rather than from the keyboard. The fn argument names a command file created by a BASIC program. The command file can contain DOS commands, data, and BASIC statements.

MAXFILES n

where n is an integer from 1 to 16, specifies the number of files that can be active at one time.

CHAIN fn [,Sn] [,Dn] [,Vn]

loads and runs an Integer BASIC program from the disk without clearing the values of any variables or arrays from memory and without closing any open files.

5.7.2 Program

MASTER

converts an initialized disk into a master disk that can operate on an Apple II system of any memory size.

Using Memory More Efficiently

117 6.1 Binary Files
118 6.1.1 Binary Addresses
118 6.1.2 An and Ln: The Memory Address Arguments
119 6.2 The Binary Commands
120 6.2.1 The BRUN Command
121 6.2.2 The BLOAD Command
122 6.2.3 The BSAVE Command
123 6.2.3.1 Example
124 6.3 DOS and the Monitor Program
125 6.3.1 The Input and Output Registers of the Monitor
126 6.3.2 The Input and Output Registers of DOS
127 6.4 Memory Usage and Entry Points
127 6.4.1 PR# and IN# in Memory
129 6.4.2 Memory Maps
129 6.4.2.1 HIMEM
132 6.4.2.2 The Values of HIMEM
132 6.4.2.3 High-Resolution Graphics
133 6.4.3 DOS Entry Points
133 6.4.3.1 Routine to Reconnect DOS
134 6.4.3.2 Binary File Memory Locations After BLOAD
134 6.4.3.3 DOS Memory Locations After BLOAD
135 6.5 Binary Command Summary

Using Memory More Efficiently

This chapter describes how to use binary programs and how they interact in memory with the Monitor program. It includes

- loading, running, and saving binary programs;
- using binary programs to read and write characters;
- controlling the input and output registers of DOS and of the Monitor.

Programmers who write in assembly and machine language use binary commands. The *Apple II Reference Manual* and the *6502 Assembler/DOS Tool Kit* (formerly called the *Apple 6502 Assembler/Editor Manual*) provide detailed information about using these languages.

6.1 Binary Files

Binary is a numeric representation in terms of powers of 2, using the digits 0 and 1.

Hexadecimal numbers are expressed in terms of powers of 16, using the digits 0 through 9 and A through F.

A **binary file** contains information stored in consecutive locations in your Apple II's memory or on a disk. The information is not expressed in text form; instead, it is a series of hexadecimal numbers from $00 to $FF (decimal 0 to 255). These hexadecimal numbers can represent numbers, letters, machine-language commands, or high-resolution graphics.

You can store values in a binary file up to the limits of your Apple II's memory. A binary file has the file type B.

6.1.1 Binary Addresses

You don't have to understand the organization of the Apple II's memory to run an existing machine-language program. But you do need to know about memory addresses to work directly with the binary information in memory.

Your computer'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 ($0001), and so on. When your computer has 48K bytes of memory, the address of the last RAM memory location is 49151 ($BFFF); for 64K, the last RAM memory location is 65535 ($FFFF).

6.1.2 An and Ln: The Memory Address Arguments

To use a binary command, you indicate the memory address by means of command arguments:

An	specifies the starting address in memory.
Ln	specifies the number of memory locations, or bytes.

Replace the lowercase n with a decimal or hexadecimal number. For example, to save a high-resolution graphics screen that is in memory locations 8192 through 16383 ($2000 through $3FFF) in a file, you specify the starting address of the information as

,A8192 or ,A$2000

and the number of bytes you're saving as

,L8192 or ,L$2000

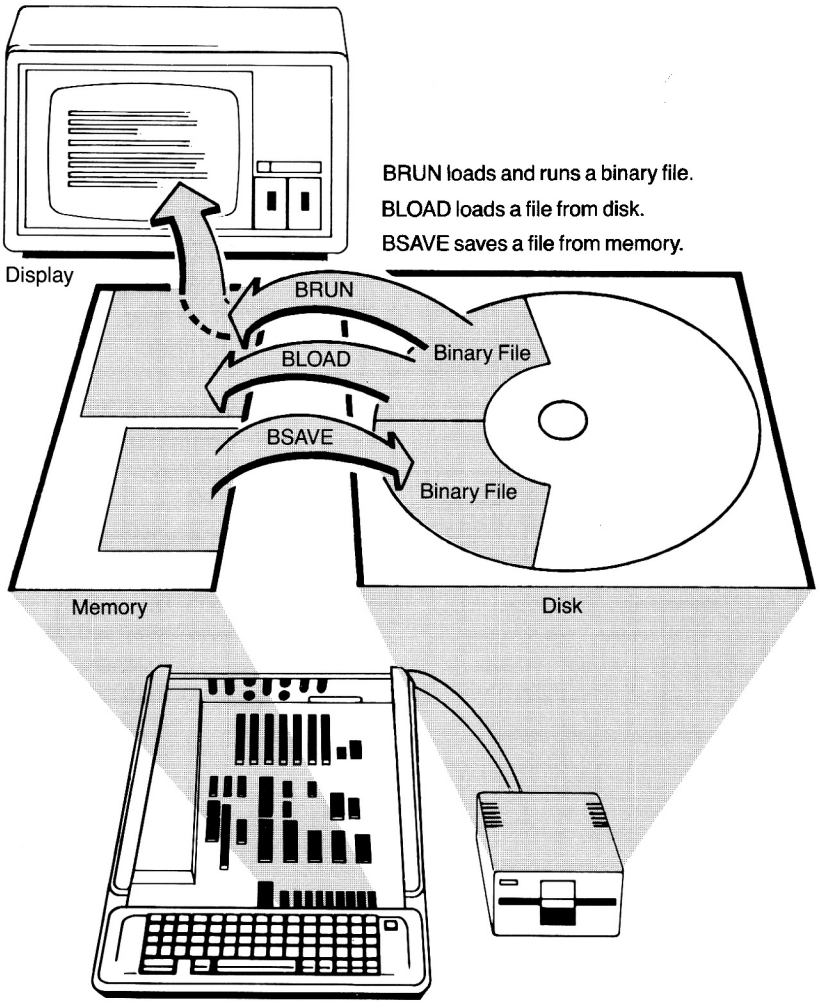
Remember: Hexadecimal numbers are preceded by $.

6.2 The Binary Commands

The DOS commands BRUN, BLOAD, and BSAVE deal with binary information in any part of the computer's memory. The B in each of these commands stands for "binary."

Binary commands transfer binary information, byte for byte, between memory and a file. The two most common uses of binary commands are running binary programs and bringing binary images into memory for display.

Figure 6-1. BRUN, BLOAD, and BSAVE



6.2.1 The BRUN Command

The BRUN command runs a binary program file (file type B). When DOS sees this command, it transfers the specified program into memory and runs the program. DOS puts the program into main memory starting at the address from which the program was saved (BSAVE) unless you specify another address. This is the command syntax:

BRUN fn [,An] [,Sn] [,Dn] [,Vn]

fn	indicates a binary file. If you use only this argument, DOS places the entire contents of the binary file fn into memory starting at the address from which it was saved with the BSAVE command.
[,An]	where n is a memory address from Ø to 65535 (decimal) or $Ø to $FFFF (hexadecimal), specifies where DOS transfers the first byte of the program.
[,Sn]	where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.
[,Dn]	where n is 1 or 2, specifies the drive of the disk containing fn. If omitted, DOS uses the default drive number.
[,Vn]	where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS ignores the volume specification.

To run a binary program named FID, type the command

```
BRUN FID
```

By the Way: DOS can't tell the difference between a binary program file and a binary data file. So it's a good idea to name binary files to indicate their contents. For example, the suffix .PIC can specify a file containing picture data.

If you accidentally run (BRUN) a data file, parts of DOS may be changed. In this case, restart DOS.

6.2.2 The BLOAD Command

With the BLOAD command, you can move a picture from a file to a graphics screen, move the binary image of any type of file into memory, or transfer a program written in machine language from a disk file to memory. The syntax is

BLOAD fn [,An] [,Sn] [,Dn] [,Vn]

fn	indicates a binary file.
[,An]	where n is a memory address from Ø to 65535 (decimal) or $Ø to $FFFF (hexadecimal), specifies where DOS transfers the first byte of the entire contents of this file. When the target address is omitted, DOS puts the file in memory starting at the address from which it was saved (BSAVE).
[,Sn]	where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.
[,Dn]	where n is 1 or 2, specifies the drive of the disk containing fn. If omitted, DOS uses the default drive number.
[,Vn]	where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS ignores the volume specification.

Warning
The BLOAD command transfers binary information from a disk file to RAM memory only.

Unlike the LOAD command, BLOAD doesn't erase programs or variables from memory unless they reside in the memory locations where the disk file will be put.

Warning
A machine-language program may not run if it is moved to a memory location other than the one from which it was saved. It's a good idea to note the address you used with the BSAVE command on the disk's label.

6.2.3 The BSAVE Command

The BSAVE command transfers binary information from any part of your computer's memory to a disk file. It allows you to transfer a machine-language program from memory to a file, to move a picture from a graphics screen to a file, or to store information in any portion of memory to a binary file.

The general form of the command is

BSAVE fn, An, Ln [,Sn] [,Dn] [,Vn]

fn	indicates a binary file.
,An	where n is a memory address from Ø to 65535 (decimal) or $Ø to $FFFF (hexadecimal), specifies the memory address from which DOS transfers the first byte of the program. This argument is required.
,Ln	where n is a number of bytes in the range of Ø to 32767 (decimal) or $Ø to $7FFF (hexadecimal), specifies the number of bytes of memory to transfer. This argument is required.
[,Sn]	where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.
[,Dn]	where n is 1 or 2, specifies the drive of the disk on which to store fn. If omitted, DOS uses the default drive number.
[,Vn]	where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS ignores the volume specification.

Reminder: On the disk's label write the address from which you save (BSAVE) a binary program.

6.2.3.1 Example

In this example you create a binary file and use it with the BSAVE, BRUN, and BLOAD commands.

The Monitor is in read-only memory and controls the computer's functions.

1. To enter the Monitor, type `CALL -151` and press RETURN.
2. Type these binary values into memory. Note that they are hexadecimal numbers.

   ```
   300: 20 B0 FE A0 00 B9 16 03 F0 06 20 ED FD
   C8 10 F5 20 B4 FE 4C D0 03 8D C1 D0 D0 CC C5
   A0 AF AF 00
   ```

   When you've entered all the values, press RETURN.
3. Ask DOS to save this data as a binary file with the name APPLEII by typing

   ```
   BSAVE APPLEII,A$300,L$20
   ```

   The An argument saves the data from location $300. The Ln argument saves data for a length of $20 bytes.
4. Run it by typing

   ```
   BRUN APPLEII
   ```

   You'll see `APPLEII` printed in inverse mode on your screen.
5. To do the same thing from BASIC, type

   ```
   BLOAD APPLEII,A$300
   CALL 768
   ```

6.3 DOS and the Monitor Program

In the previous example, you entered the Monitor program so that you could type in a binary program. This section lists many of the commands you use to enter the Monitor and to return to BASIC.

To enter the Monitor program from BASIC, type

```
CALL -151
```

You will see an asterisk (*), the prompt character for the Monitor program. DOS is still active, so you can use all the immediate-execution DOS commands. For example, the CATALOG command will display a catalog on the screen and the PR# command will run a program on a disk.

You can use several commands (n is the slot number) from the Monitor with or without DOS active:

n(CONTROL)-(P)	(Same as PR#n from the Monitor—directs output)
n(CONTROL)-(K)	(Same as IN#n from the Monitor—directs input)
(CONTROL)-(C)	(Continues BASIC—doesn't reset it)
(CONTROL)-(B)	(Enters BASIC—resets it)
*3D0G	(Enters BASIC—doesn't reset it—connects DOS again)
3D3G	(Enters BASIC—doesn't reset it—connects DOS again)
3EAG	(If DOS is in memory, connects DOS—remains in the Monitor)

To select input and output devices and to connect DOS again, you can also use POKE statements to put appropriate values directly into the Monitor registers.

6.3.1 The Input and Output Registers of the Monitor

CONTROL-K and CONTROL-P are not displayed on the screen when you type them.

In Table 6-1, nCONTROL-K and nCONTROL-P are Monitor commands.

Table 6-1. Monitor Input Register in Locations 56-57 ($38-$39)

Register contents set by	To the value	Subsequent input comes from
RESET 0CONTROL-K IN#0	−741 ($FD1B)	Monitor Input Routine from keyboard
nCONTROL-K IN#n (where n > 0)	49152 + n*256 ($Cn00)	slot n If slot n contains disk controller, boot DOS
DOS boot	Top of mem. - 8515 ($Top of mem. - $2143)	DOS

Table 6-2. Monitor Output Register in Locations 54-55 ($36-$37)

Register contents set by	To the value	Subsequent output goes to
RESET 0 CONTROL-P PR#0	−528 ($FDF0)	Monitor Output Routine to display screen
nCONTROL-P PR#n (where n > 0)	49152 + n*256 ($Cn00)	slot n If slot n contains disk controller, boot DOS
DOS boot	Top of mem. - 8575 ($Top of mem. - $217F)	DOS

6.3.2 The Input and Output Registers of DOS

Table 6-3. DOS Input Register

Register contents set by	To the value	Subsequent input comes from
DOS boot (RESET) 3D0G IN#0 PRINT D$:"IN#0" (D$ = (CONTROL)-(D))	−741 ($FD1B)	Monitor Input Routine from keyboard
IN#n PRINT D$;"IN#n" (where n > 0)	49152 + n*256 ($Cn00)	slot n If slot n contains disk controller, reboot DOS

Table 6-4. DOS Output Register

Register contents set by	To the value	Subsequent output goes to
DOS boot (RESET) 3D0G PR#0 PRINT D$;"PR#0" (D$ = (CONTROL)-(D))	−528 ($FDF0)	Monitor Output Routine to display screen
PR#n PRINT D$;"PR#n" (where 0 < n < 8)	49152 + n*256 ($Cn00)	slot n If slot n contains disk controller, reboot DOS

When DOS is operating, PR# and IN# do not affect the contents of the DOS input and output registers unless they are issued without a (CONTROL)-(D) as an instruction in a command file. For example, when a command file executes a line such as 120 PR#3, the contents of the Monitor output register are changed and DOS is partially disconnected until the next input.

No matter what input or output device is selected by the DOS input and output registers, input can also be received from the disk and output can be sent to the disk.

There are two ways to select input and output devices and to reconnect DOS: you can use the commands in Tables 6-1 through 6-4, or you can use POKE statements to put appropriate values directly into the DOS registers.

Warning

The specific memory locations of the DOS input and output registers change with different sizes of system memory and with different versions of DOS. For this reason, you need to follow a special two-step procedure to change the contents of these register locations:

For an example of this procedure, see Chapter 2 for the program for finding hidden characters.

1. Change the Monitor input and output register locations to the values you want the DOS input and output registers to contain. Either use POKE statements to place the values directly into the Monitor locations or use IN# and PR# without CONTROL-D from a program.

2. Issue a CALL 1002 (from DOS) or $3EAG (from the Monitor) to reconnect DOS through the Monitor registers. The previous contents of the Monitor input and output registers will appear in the input and output registers of DOS. This CALL can also be used to reconnect DOS whenever your program needs to disconnect DOS temporarily.

DOS does not recognize the Monitor commands nCONTROL-K and nCONTROL-P. Since these commands directly affect the Monitor input or output registers, use PR# and IN# when DOS is connected.

6.4 Memory Usage and Entry Points

This section describes how PR# and IN# use memory. Then it maps the memory areas overwritten when starting DOS, the memory addresses DOS assigns to HIMEM, and the memory areas that DOS and both BASIC dialects use.

6.4.1 PR# and IN# in Memory

CSWH and CSWL = monitor output link.

To see how PR# and IN# work in memory, examine how the Apple II normally sends and receives characters. Two memory locations, named CSWH and CSWL, store the memory address of a routine that writes characters. CSWH and CSWL are the **monitor output link**; they link the Monitor program to an output routine.

KSWH and KSWL = monitor input link.

Similarly, two other memory locations, named KSWH and KSWL, store the memory address of the routine that reads characters. KSWH and KSWL are the **monitor input link**.

COUT1 sends characters to the display screen; KEYIN reads characters from the keyboard.

The monitor output link normally contains the address of the standard output routine, COUT1, which sends characters to the display screen. The monitor input link normally contains the address of the standard input routine, KEYIN, which reads characters from the keyboard.

When DOS is *not* running and you use PR# or IN# from BASIC, the Monitor links are set to indicate the ROM on the card in the indicated slot ($Cn00 for slot n). When the computer gets a character from the keyboard or sends one to the display screen, it calls the input or output routine in the card's ROM to perform the transfer.

When DOS *is* running, the I/O links of the Monitor program contain the addresses of the DOS input and output routines instead of the standard input and output routines. DOS keeps the addresses of the Monitor's standard input and output routines in the DOS I/O links; that is, the DOS I/O links normally contain the addresses of KEYIN and COUT1.

When you use PR# or IN# with a slot number, DOS replaces the contents of the proper DOS 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, DOS places that address in the proper DOS link. Then, when the Apple II tries to write or read a character, the Monitor's output or input links point to the proper DOS routine, which is a two-part transfer:

- In part 1, DOS moves the addresses of the current I/O routines from the DOS I/O links to the Monitor's I/O links. Then DOS calls COUT1 or KEYIN to write the character to or read it from the device now selected by the Monitor's I/O links.
- In part 2, DOS reconnects itself by placing the addresses of its I/O routines into the Monitor's I/O links.

When a program issues a PR# or IN# command without a CONTROL-D, the command goes to BASIC. Then BASIC changes the values of the Monitor I/O links. Since the Monitor's links no longer point to DOS, DOS is disconnected. Because of the two-part transfer, DOS will use the old I/O routine (part 1) to transfer the next character, then it will reconnect itself (part 2) *if it isn't disconnected at both input and output registers simultaneously*.

If this should happen, reset the DOS input and output to the keyboard and display screen. If DOS is no longer in memory, restart DOS. If DOS is still in memory, use the BASIC statement CALL 1002.

By the Way: CALL executes a machine-language subroutine from a BASIC program. Control transfers to the subroutine. When it finishes, execution goes to the statement following the CALL. The CALL doesn't affect the program in memory.

6.4.2 Memory Maps

6.4.2.1 HIMEM

HIMEM = highest memory address (plus one) that Applesoft can use.

HIMEM is the highest memory address available to an Applesoft program for the storage of program statements and variables. Applesoft automatically sets HIMEM to the address of the highest writeable memory (plus one) RAM address available on your computer. Loading DOS automatically resets HIMEM to a lower value in order to protect the area of memory DOS itself occupies. Depending on the value DOS sets, the area above the HIMEM address may be available for use by DOS, high-resolution graphics, or machine-language programs.

Figure 6-2 maps main memory as it is affected by booting DOS. Figure 6-3 maps areas of memory that are used by DOS and both BASIC languages.

Locations 6912 to 16383 ($1B00 to $3FFF) are affected when starting up a master disk. They are *not* affected when starting a disk initialized on a system 32K or larger. DOS is placed directly below the highest RAM memory address that was available on the system on which the disk was prepared.

Figure 6-2. Memory Areas Overwritten When Booting DOS

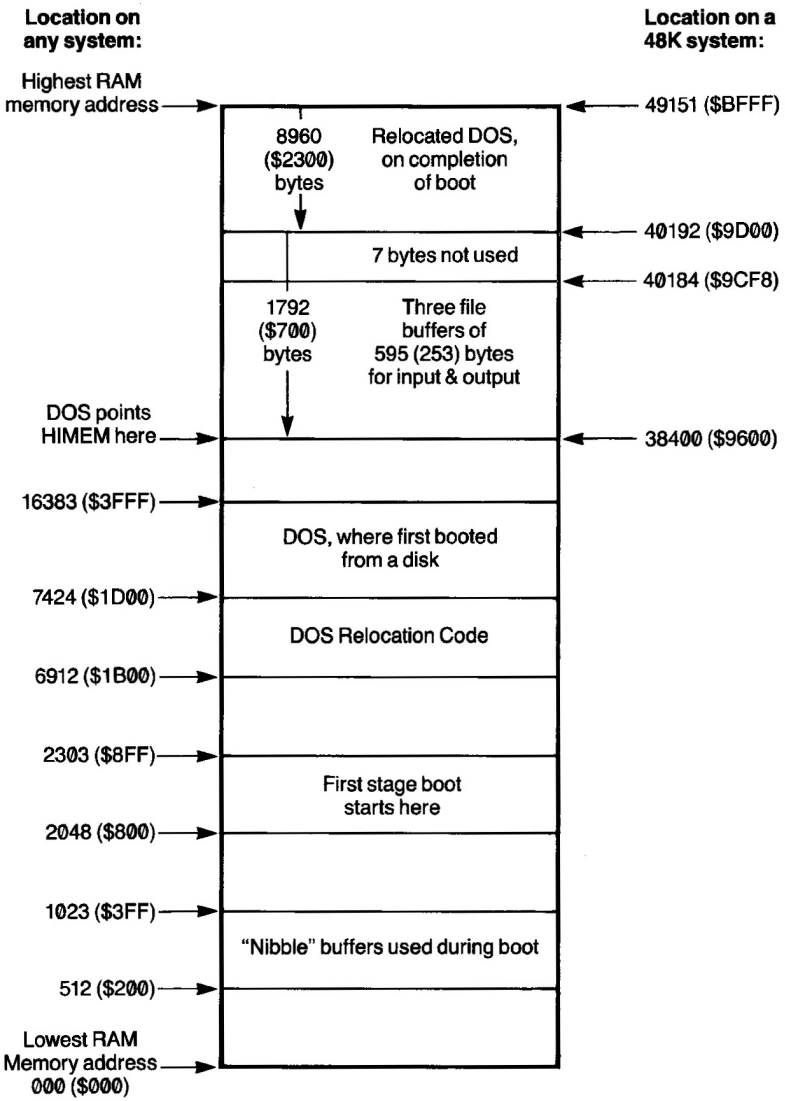
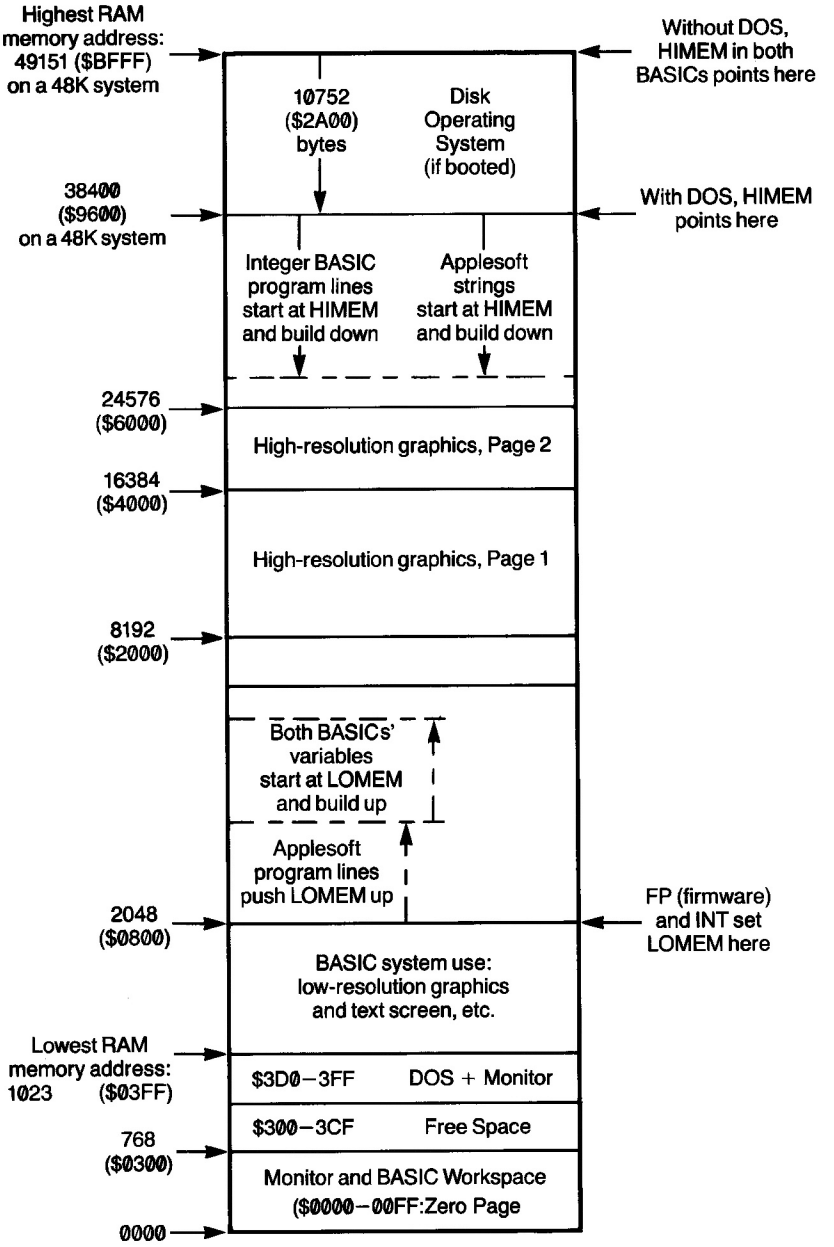


Figure 6-3. Memory Areas Used by DOS and Both BASICs



6.4.2.2 The Values of HIMEM

HIMEM is the highest memory location that is available to your program. This upper boundary is set initially by the Apple II computer.

Booting DOS sets the value of HIMEM according to the amount of memory your computer has. Table 6-5 shows these values. When your system is running Integer BASIC, the HIMEM pointer is located (low byte first, then high byte) in bytes 76-77 ($4C-$4D). When your system is running Applesoft, the HIMEM pointer is in bytes 115-116 ($73-$74).

Increasing MAXFILES moves HIMEM down 595 bytes for each file buffer you add. The *Applesoft II BASIC Programming Manual* gives the locations of other Applesoft program pointers.

Table 6-5. The Values of HIMEM

System size	Highest RAM address		HIMEM (set by DOS boot)	
	Decimal	Hexadecimal	Decimal	Hexadecimal
16K	16383	$3FFF	5632	$1600
20K	20479	$4FFF	9728	$2600
24K	24575	$5FFF	13824	$3600
32K	32767	$7FFF	22016	$5600
36K	36863	$8FFF	26112	$6600
48K (*)	49151	$BFFF	−27136	$9600 (**)

(*) These values are for systems 48K and larger.

(**) The decimal number -27136 could also be written 38400; however, Integer BASIC does not accept numbers greater than 32767.

By the Way: In Integer BASIC, memory addresses greater than 32767 must be expressed as their negative equivalents. The negative equivalent of any positive decimal address n is (n - 65536).

6.4.2.3 High-Resolution Graphics

The Apple II computer has two areas of memory for low-resolution graphics and two areas for high-resolution graphics. These areas are called **pages**. The pages are plotted on the display screen in terms of rows and columns.

Using high-resolution graphics Page 1 erases the contents of memory locations 8192 through 16383. Unless your system contains at least 32K of memory, you cannot use disks and high-resolution graphics at the same time.

Warning
HIMEM and LOMEM must be set appropriately to avoid conflict with high-resolution graphics Page 1.

Using high-resolution graphics Page 2 erases the contents of memory locations 16384 through 24575. Unless your system contains at least 32K of memory, you cannot use disks and high-resolution graphics Page 2 at the same time.

Warning
HIMEM and LOMEM must be set appropriately to avoid conflict with high-resolution graphics Page 2.

6.4.3 DOS Entry Points

This section provides DOS entry points and the ways to use them

- to reconnect DOS if it is partly overwritten.
- to find the starting address and length of a binary program brought into memory by the BLOAD command.
- to find the DOS locations of the starting address and length of the most recent program placed in memory by the BLOAD command.

6.4.3.1 Routine to Reconnect DOS

Page 3, see Figure 6-3.

The DOS pointers and subroutines found in Page 3 may be disconnected if you accidentally enter the Monitor program. Page 3 is an area of memory (addresses $0300 to $03FF) used by DOS.

To reconnect DOS, use either a CALL or the Monitor command G with the value needed for the size of your computer's memory; these values are:

System size	Decimal address (CALL)	Hexadecimal address (G)
48K *	-25153	$9DBF
32K	23999	$5DBF
16K	7615	$1DBF

* For systems 48K and larger.

The Monitor command 3D0L lists this number at the top right of the screen.

6.4.3.2 Binary File Memory Locations After BLOAD

You can find the starting address or length of a binary file after a BLOAD command. Type

```
PRINT  PEEK(low byte) + PEEK(low byte+1)*256
```

Find the appropriate low byte your Apple II uses from the list below.

System size	Starting address (low byte) Decimal	Hexadecimal	Program length (low byte) Decimal	Hexadecimal
48K*	43634	$AA72	43616	$AA60
32K	27250	$6A72	27232	$6A60
16K	10866	$2A72	10848	$2A60

* For systems 48K and larger.

6.4.3.3 DOS Memory Locations After BLOAD

This program finds the DOS locations containing the starting address and length of the program most recently brought into memory by the BLOAD command. It can be used on an Apple II with any size memory.

Memory Size: The values of H and T (lines 7 and 8) are for a 48K Apple II. Table 6-5 shows the correct values for your computer.

```
5 REM BLOAD FINDER
7 H = 38400: REM DOS-BOOT HIMEM
8 T = 49152: REM HIGHEST ADDRESS
10 D$ = CHR$ (4): REM CONTROL-D
20  PRINT D$; "BSAVE FOO, A$7777, L$77"
30  PRINT D$; "BLOAD FOO"
40  PRINT D$; "DELETE FOO"
50  FOR I = H +1792 TO T
60  IF PEEK (I) <> 119 OR PEEK (I+1) <> 0 THEN
      NEXT I
70  PRINT "LOCATIONS OF START ADDRESS: ";I;",
      ";I + 1
80  FOR I = H + 1792 TO T
90  IF PEEK (0) <> 119 OR PEEK (I+1) <> 0 THEN
      NEXT I
100 PRINT "LOCATIONS OF LENGTH: ";I;", ";I + 1
```

This program takes about two minutes to find the desired locations.

6.5 Binary Command Summary

BRUN fn [,An] [,Sn] [,Dn] [,Vn]

tranfers the contents of a binary file (file type B) on a disk to any part of memory and then executes the program. If the An (address) option is not used, the program is placed in memory starting at the address specified in the BSAVE command that originally saved the file.

BLOAD fn [,An] [,Sn] [,Dn] [,Vn]

transfers binary information from a disk to any part of memory. When the An (address) option is not used, the program is placed in memory starting at the address specified in the BSAVE command that originally saved the file.

BSAVE fn , An , Ln [,Sn] [,Dn] [,Vn]

transfers binary data from a specified portion of memory to a binary file (file type B) on a disk. The An (address) and Ln (length) arguments allow you to transfer information from any part of memory.

Appendixes

139 **A. Dealing With 13-Sector Disks**
140 A.1 Converting a 13-Sector Disk: The CONVERT13 Program
140 A.1.1 Example
143 A.1.2 The Wildcard Character
144 A.2 Running Unconverted 13-Sector Disks
144 A.2.1 Using the START13 Program
144 A.2.1.1 Example
145 A.2.2 Using the BASICS Disk
147 **B. The Storage Process**
147 B.1 Tracks and Sectors
148 B.2 Contents of File Sectors
149 B.2.1 Format of File Sectors According to File Type
149 B.2.2 The Track/Sector List
151 B.2.3 The Disk Directory
155 B.2.4 The Volume Table of Contents
157 B.2.5 The Track Bit Map
158 B.3 Track and Sector Allocation
159 B.4 Addressing Tracks and Sectors: The RWTS Subroutine
160 B.4.1 Example
162 B.4.2 Formats: I/O Block and Device-Characteristics Table
163 B.5 DOS Vectors in Page 3
165 B.6 Zero Page Use
167 **C. DOS Error Messages**
168 C.1 DOS Error Messages
169 C.2 Recovering From Errors
177 **D. Programs**
177 D.1 Programs on the SYSTEM MASTER Disk
178 D.2 Programs on the SAMPLE PROGRAMS Disk

181 **E. Summary of DOS Operating Concepts and Commands**
181 E.1 Operating Concepts
181 E.1.1 Cold Start
182 E.1.2 Warm Start
182 E.1.3 Initializing a Disk
182 E.1.4 Creating a Master Disk
183 E.1.5 Interpreting Commands
183 E.1.6 Capacity
183 E.1.7 File Types
183 5.2 Command Notation
183 E.2.1 Conventions
184 E.2.2 Syntax
184 E.2.3 Arguments
186 E.3 Command Summary
187 E.3.1 Nonprogramming Commands for Accessing Disks
190 E.3.2 Commands that Control the Programming Environment
192 E.3.3 Programming Commands for Sequential-Access Text Files
195 E.3.4 Programming Commands for Random-Access Text Files
197 E.3.5 Commands for Binary Files

Dealing With 13-Sector Disks

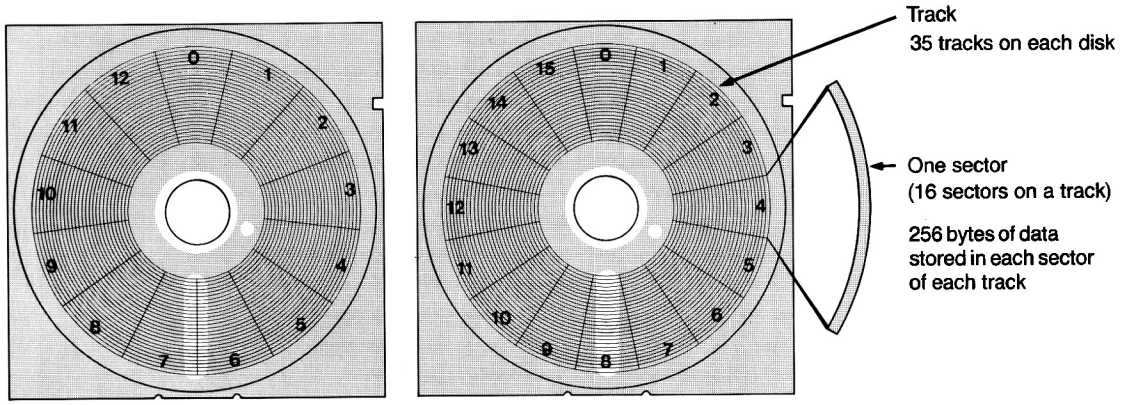
This appendix is for people who have earlier versions of DOS or who have games or other application programs that don't seem to run on their present Apple II. It tells you how to convert 13-sector disks to 16-sector format and how to use unconverted 13-sector disks with 16-sector DOS.

Earlier versions of DOS (3.1 and 3.2) organized the surface of each track on a disk into 13 sectors (see Figure A-1). But DOS 3.3 divides each track into 16 sectors, giving each disk about 20 percent more space for information.

By the Way: If you have a 13-sector system and would like to update it, install the DOS 3.3 Kit.

If you use a disk that has the wrong format for your system, DOS prints an appropriate error message, such as `UNABLE TO READ`, `UNABLE TO WRITE`, or `I/O ERROR`.

Figure A-1. 13-Sector Versus 16-Sector Disks



A.1 Converting a 13-Sector Disk: The CONVERT13 Program

The CONVERT13 program on the SYSTEM MASTER disk converts a 13-sector disk and its programs to 16-sector format. CONVERT13 reads the information from a 13-sector disk and writes the information to a disk you have initialized in the 16-sector format without changing the original information in any way.

Incidentally: The MUFFIN program, from an earlier version of DOS, performs the same function that CONVERT13 does.

A.1.1 Example

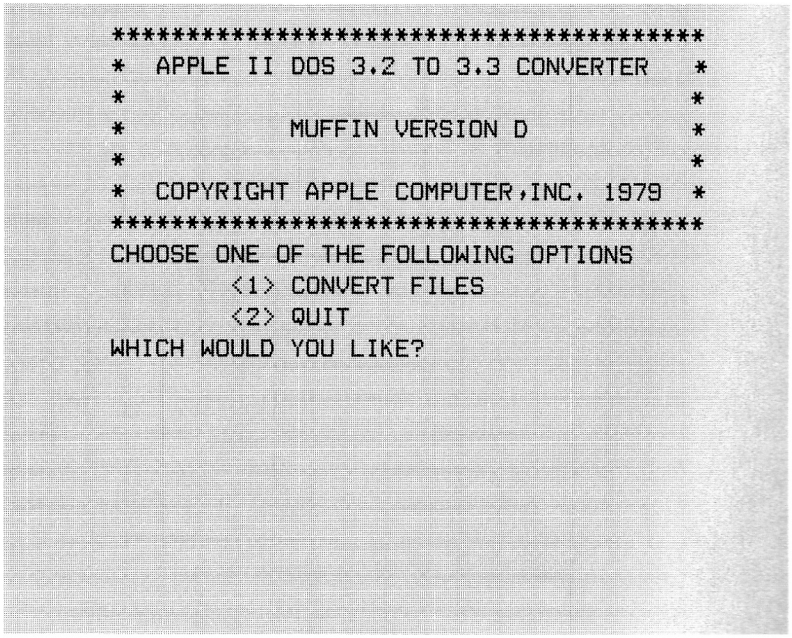
This example of CONVERT13 assumes that your Apple II has one disk drive and that your disk drive is connected to a controller card in slot 6.

1. With DOS in memory, initialize a blank disk.
2. Put the SYSTEM MASTER disk in your drive and type

   ```
   RUN CONVERT13
   ```

   The message `EXECUTING MUFFIN` appears briefly; then you'll see the CONVERT13 menu, as shown in Figure A-2.

Figure A-2. The CONVERT13 Menu



```
****************************************
*  APPLE II DOS 3.2 TO 3.3 CONVERTER   *
*                                      *
*           MUFFIN VERSION D           *
*                                      *
*  COPYRIGHT APPLE COMPUTER,INC. 1979  *
****************************************
CHOOSE ONE OF THE FOLLOWING OPTIONS
        <1> CONVERT FILES
        <2> QUIT
WHICH WOULD YOU LIKE?
```

Type

```
1
```

to indicate that you want to convert files.

3. When you see the question `SOURCE SLOT?`, type

   ```
   6
   ```

   This is the number of the slot that holds the controller card for the drive that will contain the 13-sector disk.

4. The next question is `DRIVE?`. Type

   ```
   1
   ```

   the number of your disk drive.

5. When you see `DESTINATION SLOT?`, type

   ```
   6
   ```

   Use the same slot number for the duplicate disk.

6. Answer the question `DRIVE?` by typing

   ```
   1
   ```

   the number of your disk drive.

7. Now you're asked for the name of the file to be converted: `FILENAME?`. Type

   ```
   =
   ```

Wildcard, see the next section.

The equal sign (=) is a **wildcard** representing the names of all the 13-sector files. It means you want to convert the entire contents of the 13-sector disk.

8. Before anything will be converted, CONVERT13 asks `DO YOU WANT PROMPTING?`

   For now, type

   `N` (for no).

9. When you see `INSERT DISK(S) THEN PRESS <ESC> TO RETURN TO MAIN MENU OR ANY OTHER KEY TO BEGIN`, remove the SYSTEM MASTER from the drive. From here on, you'll be working with your 13-sector disk and a newly initialized disk.

10. Insert the 13-sector disk into the drive. When you see `INSERT SOURCE DISK AND PRESS A KEY`, press (RETURN).

    CONVERT13 now finds and reads the first file on the 13-sector disk.

11. CONVERT13 displays `INSERT DESTINATION DISK AND PRESS ANY KEY` and waits for you to insert the 16-sector disk and press (RETURN).

To convert large files, you may need to swap disks several times to get the whole file transferred.

12. CONVERT13 will instruct you to insert the source disk again. Repeat this procedure, alternately reading from the source and writing to the destination disk until each 13-sector file has been converted and moved to the 16-sector disk. You'll see the message `DONE` after each file has been processed.

Two Drives: If you have more than one disk drive, specify the slot and drive numbers of the source and destination disks when the CONVERT13 program asks for them. Place the disks in the appropriate drives before the conversion begins.

When you see the message

```
INSERT DISK(S) THEN PRESS <ESC> TO RETURN TO
MAIN MENU OR ANY OTHER KEY TO BEGIN
```

you have the chance to change your mind about converting the file. If you press ESC, CONVERT13 stops and displays the menu.

If you try to convert a 13-sector file with the same name as a file already on the destination disk, you'll see:

```
          FILE [filename]
        ALREADY EXISTS.
 TYPE IN A NEW FILE NAME FOR THE COPY
 OR <RETURN> TO REPLACE EXISTING FILE
OR <CONTROL-C><RETURN> TO CANCEL COPY:
```

You can type a new name for the 13-sector file, convert the 13-sector file and have it replace the current 16-sector file, or press CONTROL-C RETURN to halt the conversion.

A.1.2 The Wildcard Character

As you saw in the example, the wildcard character (=) may be used to mean all files on the disk. It may also stand for any character or group of characters within a file name. For example, if you respond to the question FILENAME?, by typing FI=LE it converts all files whose names begin with FI and end with LE. In the same way, =TEXT converts all files whose names end with TEXT, and =*=, all files with names containing an asterisk.

When you use the wildcard, CONVERT13 asks DO YOU WANT PROMPTING? If you respond Y for yes, CONVERT13 stops after finding each file on the 13-sector disk and asks you to confirm that you want to convert that file. If you type N for no, CONVERT13 converts all the files in the wildcard group without asking you to confirm. Type Q to return to the menu.

A.2 Running Unconverted 13-Sector Disks

If your 13-sector disk is copy-protected and can't be converted, you can still use it on your 16-sector system. You can run the START13 program on the SYSTEM MASTER disk or you can start up your system with the BASICS disk.

The BASICS disk contains both BASIC languages; it is no longer part of the standard DOS package.

By the Way: The BASICS disk is comparable to the START13 program. The difference is that you *must* boot the BASICS disk to use it. In contrast, you use the RUN command to execute the START13 program directly from the SYSTEM MASTER disk.

A.2.1 Using the START13 Program

START13 is a special loader program on the SYSTEM MASTER disk that reads information from a 13-sector disk into memory. It allows you to both read and write in 13 sectors.

Incidentally: START13 is the same as the earlier BOOT13 program, which needed the BRUN command. RUN START13 is equivalent to BRUN BOOT13.

A.2.1.1 Example

The START13 example assumes that you have one disk drive connected to a disk controller card in slot 6.

1. With the SYSTEM MASTER disk in drive 1, execute the START13 program by typing

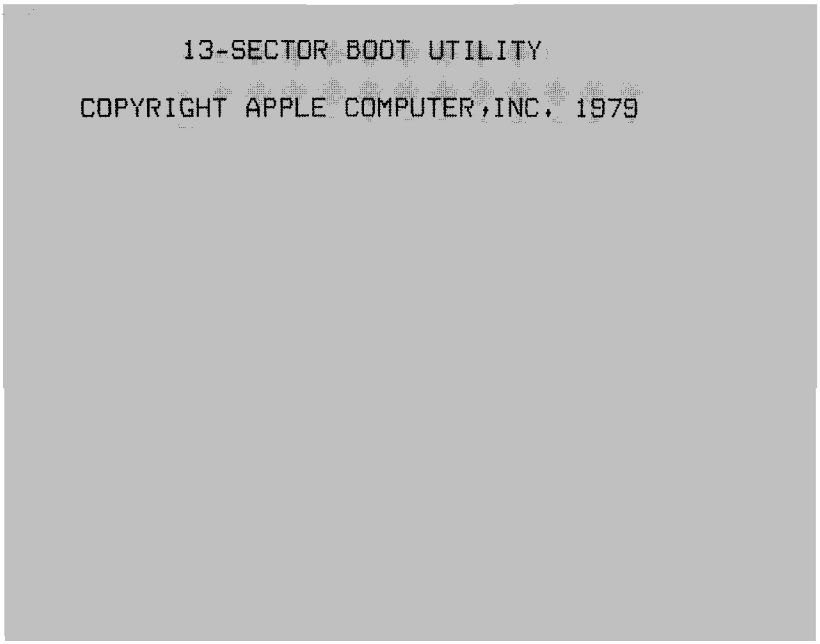
   ```
   RUN START13
   ```

   After you press (RETURN), the message `EXECUTING BOOT13` appears briefly. Then you'll see the title screen shown in Figure A-3. Remember, START13 runs BOOT13.

2. After the title screen, you'll see the question `SLOT TO BOOT FROM (DEFAULT=6)?` The program is asking what slot is connected to the drive that will hold your 13-sector disk. Put your 13-sector disk in drive 1 and press (RETURN).

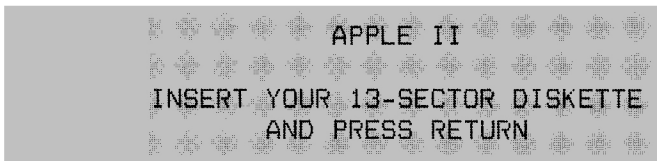
   If your 13-sector disk is a turnkey disk, the turnkey program on it will begin. So if the 13-sector disk contains Apple Writer 1.0, you'll see the menu of Apple Writer commands.

Figure A-3. The START13 Title Screen



A.2.2 Using the BASICS Disk

The BASICS disk also loads information from a 13-sector disk into memory, allowing you to both read and write in 13 sectors. To use the BASICS disk, put it in drive 1, which *must* be connected via slot 6, and turn your computer on. When you see this display on your screen



insert any 13-sector disk and press RETURN.

By the Way: The BASICS disk itself does not contain DOS.

Boot the BASICS Disk: When an earlier Apple II manual says "boot the BASICS disk," you can use either the BASICS disk or the START13 program on the SYSTEM MASTER disk.

The Storage Process

This appendix discusses the relation between tracks and sectors, and describes in detail such storage elements as the volume table of contents. It also explains how machine-language programmers can work directly with tracks and sectors and with relocatable DOS routines.

B.1 Tracks and Sectors

A dollar sign ($) indicates a hexadecimal number.

In the 16-sector DOS system, information is recorded on a disk in 35 concentric zones or bands, called **tracks**. Tracks are numbered from track $00, the outermost, through track $22, the innermost. While the disk spins, the drive's recording and reading head moves in and out to each of these 35 different tracks.

Each track on the disk is divided into 16 **sectors**, which are numbered from $0 through $F. When the drive's head is over a given track, that track's 16 sectors will pass under the head, one after the other, each time the disk spins around.

File buffer = 595 bytes: 256 for data, 256 for the track/sector list, 83 for "housekeeping."

Each sector holds up to 256 ($100) bytes of information. To store information on the disk, DOS first puts one sector's worth of the information in an area of memory called a **file buffer**. When the file buffer is full, DOS stores the information in one sector on the disk. Then DOS fills the buffer with zeros, which will be replaced by the next 256 bytes of information.

DOS always records information on the disk in 256-byte chunks, exactly filling one sector each time. If fewer than 256 bytes of data are in the buffer, the remaining bytes will be zeros.

DOS begins storing a program or text file wherever it can find an unused sector on the disk. When a sector is filled, DOS finds another free sector, perhaps on another track, and continues to record information there. This process continues until the entire file has been stored.

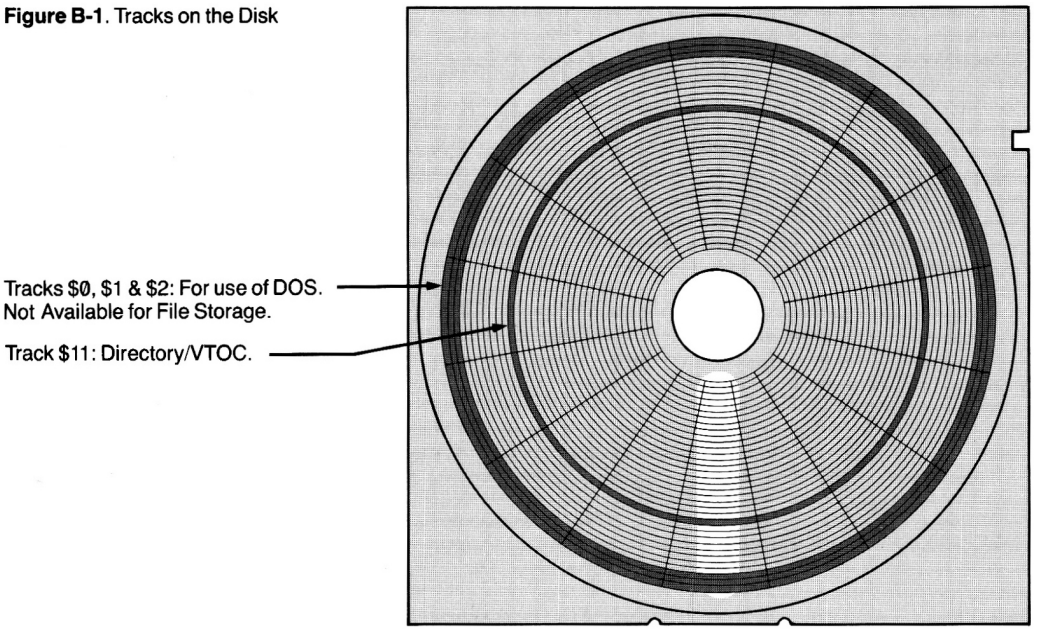
To keep track of the storage locations of the file's data, DOS maintains a list of each track and sector that the file uses. Then DOS stores that **track/sector list** in another free sector on the disk.

By the Way: When an existing file is accessed, the track/sector list is read into the second 256 bytes of the file buffer.

Finally, the file's name, file type, length in sectors, and the disk location of the file's track/sector list are recorded in a special area of track $11 called the **directory**. At this time, DOS updates the disk's **volume table of contents** to show which sectors of each track are currently in use.

Figure B-1 shows the track structure of a disk.

Figure B-1. Tracks on the Disk



B.2 Contents of File Sectors

This section describes the format of individual storage elements: file sectors, the track/sector list, the disk directory, the volume table of contents, and the track bit map.

B.2.1 Format of File Sectors According to File Type

File type is part of the "housekeeping" information in the last 83 bytes of the file buffer.

All information, regardless of file type, is stored as hexadecimal bytes. But DOS interprets the information differently depending on the file type: A, I, T, B, or R.

Table B-1 shows the format of file sectors for the different file types.

Table B-1. Format of File Sectors for Different File Types

File type	Sector	Byte	Contents of byte
A, I	1st sector	$0	Program length, low byte
		$1	Program length, high byte
		$2 through $FF	Tokenized program
	Subsequent sectors	All bytes	Tokenized program
T	All sectors	All bytes	ASCII representation of text, with high bit set: one byte/ character ($00 marks end of file)
B	1st sector	$0	Starting RAM address, low byte
		$1	Starting RAM address, high byte
		$2	Length of RAM image, low byte
		$3	Length of RAM image, high byte
		$4 through $FF	Binary data
	Subsequent sectors	All bytes	Binary data

For the sector format of file type R, see the *6502 Assembler/DOS Tool Kit*.

B.2.2 The Track/Sector List

As a file is stored on the disk, DOS makes a list of the disk locations the file uses. This **track/sector list** is stored on the disk in the same way the file itself was stored. Table B-2 shows the contents of the first sector of a track/sector list. Notice that each file is assigned a pair of bytes, one with the track and the other with the sector location; these bytes are called the track/sector pair. Notice also the link pair; these two bytes contain the location of the next portion of the track/sector list.

When the list extends beyond 122 track/sector pairs, subsequent sectors of the track/sector list are identical to the first sector (Table B-2) except that the track/sector pairs refer to subsequent groups of 122 file sectors. Also, link bytes 1 and 2 differ for each subsequent sector, although the track may be the same. When both bytes of the link are zeros, the current sector is the final portion of the track/sector list.

Table B-2. First Sector of a Track/Sector List

Byte	Contents of Byte
$Ø	Not used
$1	Link: track number where continuation of the track/sector list can be found.
$2	Link: sector number where continuation of the track/sector list can be found. (If both bytes of link = zero, no link.)
$3 through $4	Not used
$5 - $6	The position, relative to the start of the file, of the first sector of the current portion of the track/ sector list.
$7 through $B	Not used
$C	Track number of first file sector
$D	Sector number of first file sector
$E	Track number of second file sector
$F	Sector number of second file sector
$1Ø	Track number of third file sector
$11	Sector number of third file sector.
.	. .
.	. .
$FE	Track number of 122nd file sector
$FF	Sector number of 122nd file sector

Any track/sector pair that is 0/0 indicates an unassigned sector. This is usually the end of the file, although random-access text files can contain many 0/0 indicators for sectors where future records may be written.

With random-access text files, only the track/sector pairs for those sectors actually containing information appear as non-zero in the track/sector list. DOS calculates the correct position for the track/sector pair within the list; unassigned track/sector pairs are filled with zeros.

With a random-access file, when you specify the length as 127 (two records per sector) and you write only to record number 2700, DOS uses 13 disk sectors: one for the contents of record number 2700 and 12 for the sectors of the track/sector list. The contents of records 0 through 2683 may someday occupy 1342 sectors; but until those records are written, they do not use any disk space. In the track/sector list, the locations of the sectors containing records 0 to 2683 occupy 11 sectors.

B.2.3 The Disk Directory

DOS reserves track $11 of every initialized disk for the disk directory, which holds information about each file on the disk: the file's name, file type, the number of sectors the file uses (MOD 256), whether the file is locked, and the location of the file's first track/sector list. The CATALOG command displays all of this information except the location of the first track/sector list.

Table B-3 lists the contents of each sector of a disk directory.

Table B-3. One Sector of a Disk Directory

Byte	Contents of Byte
$Ø	Not used
$1	Link: Track number where continuation of the directory can be found (normally $11)
$2	Link: Sector number where continuation of the directory can be found (If both bytes of link = zero, no link.)
$3 through $A	Not used
$B through $2D	Directory entry for file 1
$2E through $5Ø	Directory entry for file 2
$51 through $73	Directory entry for file 3
$74 through $96	Directory entry for file 4
$97 through $B9	Directory entry for file 5
$BA through $DC	Directory entry for file 6
$DD through $FF	Directory entry for file 7

Directory entry, see Table B-4.

The file numbers for the seven directory entries in Figure B-4 are arbitrary. When a file is deleted, DOS marks its directory entry. The next time a file is stored, DOS overwrites a marked directory entry with the entry for the new file. Thus, while DOS originally fills the directory in the order shown, file deletions soon render this order meaningless.

The disk directory begins in track $11, sector $F. This starting sector is found in bytes 1 and 2 of the volume table of contents.

When more space is needed to store additional directory entries, sector $F is linked to sector $E. When still more space is needed, sector $E is linked to sector $D, and so on, through sector $1. This allows the directory to store entries for a maximum of 105 different files.

Each directory entry is written in the format shown in Table B-4.

Table B-4. Directory Entry for One File

Relative Byte	Contents of Byte
$Ø	Track number of the track/sector list of the file (The original value is copied into $2Ø and value of byte $Ø is changed to $FF if file is deleted.)
$1	Sector number of its track/sector list
$2	File type
$3 through $2Ø	File name
$21	Sector count: the number of disk sectors (MOD 256) occupied by the file
$22	Terminator (0)

The relative byte (column 1) of a directory entry specifies each byte within the entry, although each entry starts at a different actual byte number within the directory sector. To find the absolute sector byte corresponding to a relative byte, add the relative byte to the entry's first absolute sector byte (as shown in Table B-3).

Because only one byte is used to store a file's sector count, the maximum directory sector count is 255 ($FF). When a file exceeds 255 sectors, its sector count (as displayed by CATALOG) starts over again at sector 0. This doesn't affect the use of the file but gives an erroneous impression of the remaining space on the disk.

The eight bits of the byte that designates file type (in Table B-4, relative byte number 2 in a file's directory entry) are assigned the values shown in Table B-5.

Table B-5. Byte Indicating the File Type

Bit	CATALOG Symbol	File Type Designated
7	*	File is locked (write-protected) if this bit is 1 File is unlocked (not protected) if this bit is 0
6-5		Expansion type for future use (normally zero)
4	R	Relocatable EDASM file when this bit is 1
3	S	Expansion type for future use (normally zero)
2	B	Binary file when this bit is 1
1	A	Applesoft BASIC file when this bit is 1
0	I	Integer BASIC file when this bit is 1
6-0	T	Text file when bits 6 through 0 are all zero

The file type is determined by a 1 in one of the bits 6 through 0. If bits 6 through 0 contain only zeros, the file type defaults to a text file.

The byte that designates the file type can take on the values indicated in Table B-6.

Table B-6. Locked/Unlocked Values of the File Type Byte

File Type	Value of Type Byte	
	File unlocked	File locked
T	$0	$80
I	$1	$81
A	$2	$82
B	$4	$84
S	$8	$88
R	$10	$90

B.2.4 The Volume Table of Contents

Sector $0 of track $11 contains the disk's volume table of contents (VTOC). The VTOC stores the information shown in Table B-7.

VTOC = volume table of contents.

Table B-7. Volume Table of Contents (Track $11, Sector $0)

Byte	Value	Description
$0	$2	Variable value; not used
$1	$11	Track number of first directory sector
$2	$0F	Sector number of first directory sector
$3	$3	DOS release number
$4	$0	Not used
$5	$0	Not used
$6	$1 - $FE	Disk volume number (default: $FE)
$7 - $26	$0	Not used
$27	$7A	Maximum number of track/sector pairs possible in each sector of track/sector list
$28 - $2F	$0	Not used

Byte	Value	Description
$30	$FF	Last track allocated
$31	$FF	Direction of allocation (+ = higher track)
$32	$00	Unused
$33	$00	Unused
$34	$23	Number of tracks per disk
$35	$0F	Number of sectors per track
$36	$00	Number of bytes per sector, low byte
$37	$01	Number of bytes per sector, high byte
$38 - $3B	$0	Track 0 bit map (Not available)
$3C - $3F	$0	Track 1 bit map (Available only if
$40 - $43	$0	Track 2 bit map VTOC is altered)
$44 and $45	?	Track 3 bit map
$46 and $47	$0	Track 3 bit map
$48 and $49	?	Track 4 bit map
$4A and $4B	$0	Track 4 bit map (Unused)
.	.	.
.	.	.
$78 and $79	?	Track $10 bit map
$7A and $7B	$0	Track $10 bit map
$7C - $7D	$0	Track $11 bit map (Directory & VTOC)
$7E - $7F	$0	Track $11 bit map (Unused)
$80 and $81	?	Track $12 bit map (Unused)
$82 and $83	$0	Track $12 bit map
.	.	.
.	.	.
$C0 and $C1	?	Track $22 bit map
$C2 and $C3	$0	Track $22 bit map (Unused)
$C4 - $FF	$0	Not used

B.2.5 The Track Bit Map

Starting in byte $38 of the VTOC (Table B-7), subsequent 4-byte groups each contain the track bit map for one of the 35 tracks on the disk. The arrangement of 1-bits and 0-bits within the track bit map shows DOS which sectors of that track are currently in use and which are free. The bit map for each track uses the format shown in Table B-8.

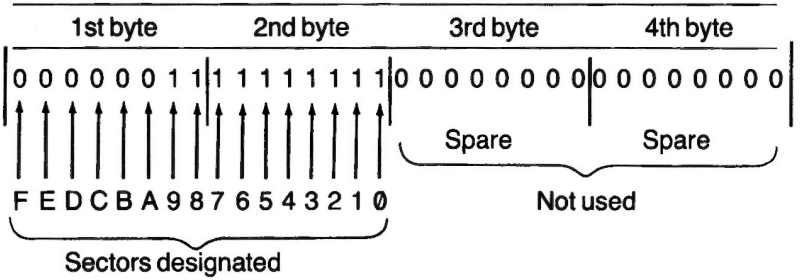
Table B-8. Track Bit Map for One Disk Track

Byte	Bit	Designated Sector	Byte	Bit	Designated Sector
1st	7	$F	2nd	7	$7
	6	$E		6	$6
	5	$D		5	$5
	4	$C		4	$4
	3	$B		3	$3
	2	$A		2	$2
	1	$9		1	$1
	0	$8		0	$0
			3rd & 4th	All	Spare

When a bit in the track bit map is 1, the sector that corresponds to that bit is free. When a bit in the map is 0, the corresponding sector is currently in use. Bits marked "Spare" contain 0; these bits are not used.

Table B-9 shows the track bit map for a typical track. The sector numbers are hexadecimal.

Table B-9. Typical Track Bit Map



1 = Free sector (assuming the corresponding bit of the mask, VTOC bytes $30 and $31, is also 1)

0 = Sector in use

When you store a file using WRITE, SAVE, or BSAVE, an entire track is allocated to the file if possible, and the track's bit map shows the entire track in use. Then, when the file is closed and the VTOC is updated, those sectors not actually used are again designated as free in the track bit map.

To Release Sectors: The sectors actually used for a file's information, however, can only be "set free" when that file name is deleted from the directory. For example, you have a disk containing a 100-sector BASIC file named BIG. If you now save a 2-sector file *with the same name to the same disk*, you will overwrite the older BIG file. But the catalog will continue to list your new, 2-sector file as having 100 sectors. To free up the 98 sectors your file no longer needs, issue this series of commands:

```
LOAD BIG
DELETE BIG
SAVE BIG
```

Release sectors no longer needed by binary files with a similar command series.

If you want to save the data and release sectors no longer needed by a text file, you have to read each of the file's fields into memory. If you store all the fields in an array, you can then delete the original file before you write each record back to the disk with the original file name.

Another way to free up text-file sectors is to read each field into memory and then immediately write the field back to the disk under a new file name. After you read and write the last field, delete the original file.

B.3 Track and Sector Allocation

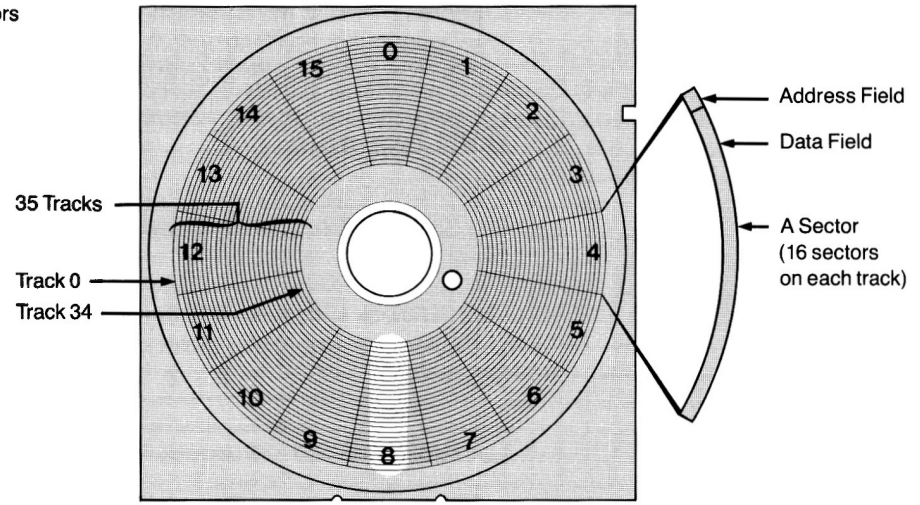
Each disk contains 35 tracks, three of which are reserved for DOS and one of which is reserved for the directory, leaving 31 tracks for the user. Each track contains 16 sectors, so 31*16 or 496 sectors are available to you.

On a freshly initialized disk, sectors are filled starting with sector $F and working back to sector $0. Tracks are first filled starting with track $12 (just inside the track for the directory/VTOC) and proceeding inward to track $22 (the innermost). When track $22 is full, tracks are then filled starting with track $10 (just outside the directory/VTOC track) and working outward to track $3 (the outermost track available to the user).

B.4 Addressing Tracks and Sectors: The RWTS Subroutine

As shown in Figure B-2, each sector has an address field and a data field. The address field contains information concerning which track the head is on, which sector is about to spin past the head, and the volume number of the disk. The data field contains the actual 256 bytes of data that are stored on the sector.

Figure B-2. Tracks and Sectors



DOS determines tracks and sectors for you, however, machine-language programmers can assign track and sector by calling the RWTS (Read or Write a Track and Sector) subroutine of DOS from a machine-language program.

The RWTS subroutine writes information to or reads information from a particular track and sector on a disk. To use the RWTS, you must first create and store in memory an I/O block, a device-characteristics table, and a controlling subroutine.

I/O block, see Table B-10.

- The *I/O block* tells the RWTS subroutine which slot and drive number to use and specifies volume number, the track and sector to access, and whether to read or write.

Device-characteristics table, see Table B-11.

- The *device-characteristics table* describes the device—in this case, a disk drive.
- The *controlling subroutine* stores the address of the starting location of the I/O block into the A and Y registers. The A register contains the high byte and the Y register, the low byte of the address. Finally, the controlling subroutine transfers control (with an assembler JSR instruction) to the starting address of the RWTS subroutine, location $3D9.

B.4.1 Example

The following sample I/O block, device-characteristics table, and controlling subroutine are loaded into memory beginning at location $C00.

This I/O block specifies slot 6, drive 1 for input and output. It writes 256 bytes of memory starting at location $C0A, on track $12, sector $06.

Location	Code	Purpose
$C0A	01	I/O block type indicator, must be $01
$C0B	60	Slot number times 16
$C0C	01	Disk drive number
$C0D	00	Expected volume number
$C0E	12	Track number
$C0F	06	Sector number
$C10	20	Low-order byte of starting address of device-characteristics table
$C11	0C	High-order byte of starting address of device-characteristics table
$C12	00	Low-order byte of starting address of data buffer
$C13	20	High-order byte of starting address of data buffer
$C14	00	Unused
$C15	00	Unused
$C16	02	Command code, $02 = write
$C17	00	Error code
$C18	00	Actual volume number
$C19	60	Previous slot number accessed
$C1A	01	Previous driver number accessed

The following controlling subroutine loads the A and Y registers with the address of the starting location of the I/O block; then it transfers execution to the RWTS subroutine.

```
$C00  A9  0C     LDA  #$0C   Load A register with $0C
$C02- A0  0A     LDY  #$0A   Load Y register with $0A
$C04- 20  D9 03  JSR  $03D9  Jump to the RWTS subroutine
$C07- 60         RTS
$C08- 00         BRK
```

Store the device-characteristics table in location $C20 following the I/O block. (In this example locations $C10 and $C11 of the I/O block above point to this starting address.) The device-characteristics table contains the following code:

Location	Code	Purpose
$C20	00	contains the device type code
$C21	01	contains the number of phases per track
$C22	EF	contains the time count
$C23	D8	contains the time count

After you store an I/O block at $C0A, a device-characteristics table at $C20, and a controlling subroutine at $C00, run the entire routine by issuing either of these two commands:

```
C00G                    or              CALL 3072
```

B.4.2 Formats: I/O Block and Device-Characteristics Table

Table B-10 shows the format of an I/O block.

Table B-10. Format of an I/O Block

Byte Number	Name	Purpose
01	1BTYPE	identifies the type of I/O block. Only type code $01 is currently defined.
02	1BSLOT	contains the number-times-16 of the slot that holds the drive's controller card. For example, to access slot 6, store the value $60 in this location.
03	IBDRVN	contains the number (either $01 or $02) of the drive to access.
04	IBVOL	contains the volume number to access; $00 will match all volume numbers.
05	IBTRK	contains the number of the track to access. It must be within the range $00 to $22 (0 to 34).
06	IBSECT	contains the number of the sector to access. It must be within the range $00 to $0F (0 to 15).
07-08	IBBUFP	contains the starting address of the data buffer (256 bytes of memory) that the RWTS uses. When you write to a disk, the data is written from the buffer to the disk. When you read from a disk, data is put in the buffer. The RWTS reads data only in chunks of 256 bytes.
0B-0C		Unused
0D	IBCMD	specifies the operation that the RWTS will perform. The values that can be stored in byte $0D (13) are: $00—starts the drive and positions the head. $01—reads the 256 bytes stored at the specified track and sector and stores them in the data buffer. $02—writes the next 256 bytes from the buffer to the specified track and sector. $04—formats the disk, writing self-synchronizing nibbles on every track and sector. Because the entire disk is formatted, the values in bytes $05 and $06 are ignored. The entire formatted disk is available for use; nothing, not even DOS, is stored on the disk until you put it there.
0E	IBSTAT	contains the code number for errors: $00—no error. $10—the disk is write-protected. $20—the volume number found differed from the number specified in byte $04. $40—drive error.
0F	IBSMOD	contains the actual volume number.
10	IOBPSN	contains the number-times-16 of the slot last accessed. For example, if you previously accessed a drive in slot 5, store the value $05 here. If there is no controller in the specified slot, the Apple II will hang.
11	IOBPDN	contains the number of the drive last accessed, either $01 or $02.

Table B-11 shows the format of a device-characteristics table.

Table B-11. Format of a Device-Characteristics Table

Byte Number	Name	Purpose
01	DEVTPC	Device type code tells what type of device to use. If it's a drive, store $00 in this byte.
02	PPTC	store $01 here.
03-04	MONTC	store the complement of the motor-on time count in 100 micro-second intervals. If the device is a drive, put $EF in byte 3 and $D8 in byte 4.

B.5 DOS Vectors in Page 3

DOS occupies 10.5K of RAM. In addition, DOS uses a group of vectors in Page 3 (locations $3D0 through $3FF, below the primary text-page 1).

Memory areas used by DOS and both BASICs, see Figure 6-3.

Since DOS can be located in different areas of memory depending on the memory size of system and whether or not DOS came in from a master disk, the addresses of callable DOS routines will change from system to system and from disk to disk. When you place the addresses of DOS routines into a vector at a fixed location in Page 3 of memory, you are no longer dependent on the varying DOS locations in high memory.

Table B-13 lists the Page-3 addresses and their contents that DOS 3.3 uses.

You can also use the addresses in Page 3 for locating DOS subroutines from versions prior to 3.3. Earlier versions of DOS use locations $300 through $3CF to load the BOOT1 program; DOS 3.3 uses those locations as a data buffer and a disk code translate table.

Table B-12. DOS 3.3 Vectors
($3D0-$3FF)

Address	Contents
$3D0	A JMP to the DOS warm-start routine that reconnects DOS, keeping the current program and MAXFILES setting.
$3D3	A JMP to the DOS cold-start routine that reconnects DOS as if it were rebooted. The current program is lost, MAXFILES returns to the default number, and HIMEM is reset.
$3D6	A JMP to the DOS file-manager subroutine so your assembly-language program can call it.
$3D9	A JMP to the RWTS routine so your assembly-language programs can call it.
$3DC	A subroutine that locates the input-parameter list so your program can create one before using the file manager.
$3E3	A subroutine that locates the IOB parameter list so your program can create one before calling the RWTS.
$3EA	A JMP to the DOS subroutine that re-establishes the I/O vectors of DOS.
$3EF	A JMP to the routine that handles a BRK machine-language instruction (Autostart ROM only). Normally this vector contains the address of the Monitor ROM subroutine that displays registers.
$3F2-$3F3	The low-byte and high-byte address of the routine that handles RESET for the Autostart ROM. Usually the DOS restart address is here.
$3F4	This is the **cold-start** byte. It contains the complement of the RESET address to distinguish between cold start and (RESET): If a cold start occurs, the Autostart ROM ignores the address at $3F2 (above) and attempts to boot a disk. To prevent this when you change $3F2 to handle your own resets, store the new value as an EOR with $A5 and then store the result in the cold-start byte.
$3F5	A JMP to a machine-language routine that is called when the ampersand (&) is used in an Applesoft statement.
$3F8	A JMP to a machine-language routine that is called when the Monitor reads a (CONTROL)-(Y).
$3FB	A JMP to a machine-language routine that ROM calls when a nonmaskable interrupt occurs.
$3FE-$3FF	The low-byte and high-byte address of a routine that ROM calls when a maskable interrupt occurs.

B.6 Zero Page Use

The **zero page**, addresses $0 through $FF, is the Monitor and BASIC workspace. Machine-language programmers use this memory area for indexed indirect commands and special functions that need highly condensed code. Table B-13 shows how DOS uses the zero page.

Table B-13. DOS Zero Page Use

Byte	Use
$24	Cursor horizontal (DOS)
$26-$27	Read buffer used by boot routine
	RWTS workspace
$2A	RWTS workspace
$2B	Boot slot times 16
	RWTS workspace
$2C	Checksum
$2D	Sector number (RWTS)
$2E	Track number (RWTS)
$2F	Volume number (RWTS)
	RWTS workspace
$33	Prompt character (DOS)
$35	Drive number in high bit (RWTS)
$36-$37	CSWL,CSWH (DOS)
$38-$39	KSWL,KSWH (DOS)
$3C	RWTS workspace
	Device-characteristics table address (RWTS)
$3D	Sector number (BOOT)
	Device-characteristics table address (RWTS)
$3E-$3F	Address of ROM sector-read subroutine (BOOT)
	RWTS buffer address
$40-$41	DOS image address (BOOT)
	DOS file buffer address
$41	Volume number used in INIT
$42-$43	DOS buffer address
$44	Track number used in INIT
$45	Sync byte used in INIT
$46-$47	RWTS workspace
$48-$49	IOB address pointer (RWTS)
$4A-$4B	LOMEM address for Integer BASIC (DOS)
$4C-$4D	HIMEM address for Integer BASIC (DOS)
$67-$68	Pointer to beginning of Applesoft program
$69-$6A	Pointer to start of Applesoft variable space
$6F-$70	Pointer to start of Applesoft string storage
$73-$74	Highest location in memory, plus one, for Applesoft
$76	Applesoft line number being executed
$AF-$B0	Pointer to end of Applesoft program
$CA-$CB	Pointer to beginning of Integer BASIC program
$CC-$CD	Pointer to end of Integer BASIC variable space
$D6	Applesoft write-protect flag (DOS)
$D8-$D9	Integer BASIC line number (DOS)
	Applesoft ONERR (DOS)

DOS Error Messages

This appendix discusses only DOS errors. An error message alerts you to a problem and indicates that the command you have issued is incompatible with DOS. The form of an error message tells you whether it comes from DOS or from one of the BASIC languages.

Form	Message Sent By
SYNTAX ERROR	DOS
?SYNTAX ERROR	Applesoft
*** SYNTAX ERR	Integer BASIC

Errors are always indicated by code numbers from 1 to 15 in Applesoft ONERR routines.

Section C.1 lists the DOS error messages. Section C.2 discusses each message and suggests ways to fix the problem.

C.1 DOS Error Messages

Table C-1 lists DOS error messages in alphabetic order; Table C-2, in numeric order.

Table C-1. DOS Error Messages

Message	Code	Meaning
DISK FULL	9	Too many files on disk
END OF DATA	5	Reading beyond end of text file
FILE LOCKED	10	Attempt to overwrite a locked file
FILE NOT FOUND	6	File misspelled or not on disk
FILE TYPE MISMATCH	13	Disk file doesn't match command
I/O ERROR	8	Door open or disk not initialized
LANGUAGE NOT AVAILABLE	1	Requested language is not there
NO BUFFERS AVAILABLE	12	Too many text files are open
NOT DIRECT COMMAND	15	Command must be in a program
PROGRAM TOO LARGE	14	Insufficient memory available
RANGE ERROR	2,3	Command parameter too large
SYNTAX ERROR	11	Bad file name, parameter, or comma
VOLUME MISMATCH	7	Wrong volume parameter
WRITE PROTECTED	4	write-protect tab on disk

Table C-2. DOS Error Codes

Code	Message	Meaning
1	LANGUAGE NOT AVAILABLE	Requested language is not there
2,3	RANGE ERROR	Command parameter too large
4	WRITE PROTECTED	Write-protect tab on disk
5	END OF DATA	Reading beyond end of text file
6	FILE NOT FOUND	File misspelled or not on disk
7	VOLUME MISMATCH	Wrong volume parameter
8	I/O ERROR	Door open or disk not initialized
9	DISK FULL	Too many files on disk
10	FILE LOCKED	Attempt to overwrite a locked file
11	SYNTAX ERROR	Bad file name, argument, or comma
12	NO BUFFERS AVAILABLE	Too many text files are open
13	FILE TYPE MISMATCH	Disk file doesn't match command
14	PROGRAM TOO LARGE	Insufficient memory available
15	NOT DIRECT COMMAND	Command must be in a program

C.2 Recovering From Errors

The following DOS error messages are listed in numeric order.

1 LANGUAGE NOT AVAILABLE

The commands FP, INT, LOAD, or RUN may initiate a language search. LANGUAGE NOT AVAILABLE means that DOS cannot find the BASIC your program needs.

You'll see this message after you issue a command that requests Applesoft from the disk in the current drive and that disk does not contain the APPLESOFT and FPBASIC programs.

You'll see this message after you issue a command that requests Integer BASIC and Integer BASIC is not in your computer.

So long as your computer has enough memory to use both Applesoft and Integer, put a disk that contains the FPBASIC, INTBASIC, HELLO, and LOADER.OBJ0 programs into a drive and issue the command again. The SYSTEM MASTER disk has these files.

When you request Applesoft, DOS first looks for the language in ROM, on an Applesoft firmware ROM card, or on the Language Card. If Applesoft is not there, DOS looks in RAM. When Applesoft is not in RAM, DOS looks on the disk in the current disk drive (the most recent values of the S and D arguments). When you request Integer BASIC, DOS looks for that language in ROM or RAM.

2 or 3 RANGE ERROR

RANGE ERROR means that the value of a command argument is too large or too small. Table C-3 shows the values that can be used with each argument.

Table C-3. Minimum and Maximum Values of Arguments

		Argument	Minimum	Maximum
All Files:	Slot	S	1	7
	Drive	D	1	2
	Volume	V	0	254
Sequential-Access Text Files:	Byte	B	0	32767
	Relative Field	R	0	32767
	Absolute Field (EXEC)	R	0	32767
Random-Access Text Files:	Record Length	L	1	32767
	Record Number	R	0	32767
Binary Files:	Starting Address	A	0	65535
	Number of Bytes	L	1	32767
DOS Commands:	PR#n	n	0	16*
	IN#n	n	0	16*
	MAXFILES n	n	1	16

* Issuing PR# and IN# with arguments of 8 to 16 can have unpredictable results.

A value beyond the allowable range does not always cause the RANGE ERROR message. Any DOS command with a value less than 0 or greater than 65535 may return a SYNTAX ERROR message.

4 WRITE PROTECTED

WRITE PROTECTED means that DOS is unable to save, write, or delete information on a write-protected disk. Either the disk has no write-enable notch or the notch is covered with a write-protect tab.

- If there is a write-protect tab over the disk's notch, remove the tab and issue the command again.
- If you receive this message while running the COPY program, you may have inserted the disk into the drive incorrectly. Check the disk's position in the drive.
- Choose another disk to save your file on. A disk without a write-enable notch is permanently protected from changes and deletions.

5 END OF DATA

END OF DATA means that your program tried to retrieve information from an area of a text file that contains no data.

Any byte beyond the last field in a sequential-access text file or beyond the last field of each record in a random-access text file contains the value 0, or $00, the ASCII code for a null character. Any command that tries to read a null character produces the END OF DATA message.

You'll see an END OF DATA message when

- a B (byte) argument is too large. In sequential access, do not specify a byte beyond the last carriage-return character in the file. In random access, do not specify a byte beyond the last carriage-return character in the currently selected record.

Remember: The first byte in a file or a record is byte 0.

- an R (relative-field position) argument in a POSITION command is too large. In sequential access, do not specify a field beyond the last existing field in the file. In random access, do not specify a field beyond the last existing field in the currently selected record.

Remember: The POSITION command moves *forward* in the file; the command's R argument specifies a field in the file relative to the current position.

- an R argument in an EXEC command is too large. It may specify a line beyond the end of the command file.
- an R argument in a READ command specifies an empty random-access record. Before you can read from a particular record in a random-access file, write some information into that record.
- an L argument of an OPEN preceding a READ differs from an L argument of an OPEN preceding a WRITE.

You'll see an END OF DATA message after an INPUT or a GET command when your program has

- too many successive INPUTs or an INPUT with too many variables. Each INPUT or INPUT variable reads an additional, adjacent field into the computer.

- too many successive GET statements. Each GET reads one additional adjacent byte or character into the computer.

6 FILE NOT FOUND

`FILE NOT FOUND` means that you specified the name of a file that is not on the disk that you are currently using.

Check the disk's catalog:

- You may have misspelled the file name by typing it incorrectly or by omitting the comma that separates the file name from a following argument.
- The file may have been accidentally deleted or may be on another disk.

If you see a `FILE NOT FOUND` message each time you start a disk, you must tell DOS the name of a greeting program on that disk. If you have no files on the disk that you want to save, you can initialize the disk again. If you can remember the name of one of the files on the disk, run the MASTER program to rename the greeting program with that file name.

7 VOLUME MISMATCH

`VOLUME MISMATCH` means that the volume (V) number used in a DOS command differs from the volume number assigned to the disk when it was initialized. Use the CATALOG command to check the volume number of the disk.

8 I/O ERROR

`I/O ERROR` means that DOS was unable to store information on a disk or to retrieve information from a disk. (DOS tries 96 times.)

Check your disk:

- It must be correctly inserted in the selected or default disk drive and the drive door must be closed.
- It must be initialized.
- If it is a 13-sector disk, see Appendix A.

Check your command arguments:

- A D argument may have specified a disk drive that doesn't exist in your system and a nonexistent drive is now the default. Specify the correct D argument with the next DOS command.
- The S argument specified a slot that doesn't contain a disk controller card. An erroneous slot is now the default, and DOS assumes that the disk that isn't connected to the slot is still running. Even if the next DOS command specifies the right slot, DOS will wait in limbo forever for the nonexistent disk to stop running the last command. If you have no program in memory, restart DOS. To recover with your program intact, first press (RESET) when the system hangs; then, to change the default, type `CATALOG, Sn`, where n is the correct slot number.

With a VERIFY command, check to find out if the file was stored correctly on the disk. Or, if the file is still in memory, try storing it on the current disk again or on a different disk.

9 DISK FULL

`DISK FULL` means that DOS tried to store information on a disk on which no space was available. In this situation, DOS closes all files and saves all the information that it can.

If you get a `DISK FULL` message, you can delete a file or two on the current disk before trying to save the information in memory. Or save it on a disk that has more room.

By the Way: If you receive this message and try again to save to the full disk, the sector length of one of the existing files in that disk's catalog will be set to 0. Despite the odd appearance of that catalog entry, the existing file itself will not be damaged in any way.

10 FILE LOCKED

`FILE LOCKED` means you tried to save, write to, change, append, or delete a locked file. Check the catalog: the name of a locked file is preceded by an asterisk (*). To unlock the file, use the UNLOCK command.

11 SYNTAX ERROR

`SYNTAX ERROR` means that you or your program issued a DOS command with an incorrect value or incorrect separator (comma or space). You will also see this message when the command lacks a required argument. Check the command's syntax in this manual.

12 NO BUFFERS AVAILABLE

Each open file and each DOS command (except PR#, IN#, and MAXFILES) requires a file buffer. `NO BUFFERS AVAILABLE` means that you or your program tried to open one more file or issued one more DOS command than there were buffers available in memory.

Issue the CLOSE command to release file buffers or issue the MAXFILES to increase the number of file buffers.

13 FILE TYPE MISMATCH

`FILE TYPE MISMATCH` means that a DOS command specified a file with a file type that is inappropriate to the present command.

Use the CATALOG command to look at the file type of the file on the disk. Then look at Table C-4 to make sure that the command you are using is legal with that file type.

If you're sure that the command is correct, use a file name that is not now on the disk, use a different disk, or rename or delete the existing file.

Table C-4. Types of Files According to Command

Command	Legal File Type
LOAD, RUN, SAVE	A or I
CHAIN	I
OPEN, READ, WRITE, CLOSE APPEND, POSITION, EXEC	T
BLOAD, BRUN, BSAVE	B

14 PROGRAM TOO LARGE

PROGRAM TOO LARGE means that a DOS command tried to load a disk file and found insufficient space in main memory for the entire file.

If you are in immediate-execution mode, issue either an FP or INT command, whichever is appropriate to your program.

Use MAXFILES to decrease the number of file buffers that are available to your BASIC program.

HIMEM is the highest memory location available to your program. For your system's maximum HIMEM with DOS and three file buffers, see Appendix B.

Often a previous program set HIMEM or LOMEM to values that will not allow the loading of your next program. Use POKE to place the correct values into the HIMEM or LOMEM locations.

By the Way: To determine whether or not a program will fit into memory, DOS looks only at the number of disk sectors the program occupies. Usually, a program will not completely fill the last file sector (256 bytes), but DOS ignores this. DOS compares the high-order byte of LOMEM (Integer BASIC) or HIMEM (Applesoft) with the high-order byte of the projected end-of-program location. Thus, programs that should fit into memory and that leave less than 256 bytes free after loading cause this message. Sometimes you can correct this before loading the program by moving HIMEM or LOMEM slightly to change the high-order byte.

15 NOT DIRECT COMMAND

NOT DIRECT COMMAND means that you tried to use an APPEND, OPEN, POSITION, READ, or WRITE command in immediate execution. These commands can be used only from PRINT statements in a program.

This message can also occur when a program is stopped and restarted. Repair the error by typing POKE 51; 0:CONT.

Programs

D.1 Programs on the SYSTEM MASTER Disk

These are the programs on the SYSTEM MASTER disk:

Name	Function
HELLO	an Applesoft greeting program that DOS runs automatically if the Applesoft language is available.
APPLESOFT	a greeting program in Integer BASIC that DOS runs automatically when the Applesoft language is not available.
BOOT13	a binary program that loads 13-sector disks created under earlier versions of DOS (before version 3.3).
CHAIN	a binary program that loads and runs a second program without erasing from memory the variables and arrays of the first. Both chained programs must be Applesoft BASIC.
CONVERT13	an Applesoft BASIC program that runs the MUFFIN program to convert 13-sector disks to 16-sector format.
COPY	the copying program to use when you are running Integer BASIC.
COPY.OBJ0	a machine-language routine used by COPY and COPYA.
COPYA	the copying program to use when you are running Applesoft BASIC.
FID	a binary program that performs several support functions for DOS.
FILEM	an Applesoft program that runs FID.
FPBASIC	the Applesoft BASIC language on disk, in binary.
INTBASIC	the Integer BASIC language on disk, in binary.
LOADER.OBJ0	a machine-language program loaded by HELLO on the SYSTEM MASTER disk. LOADER.OBJ0 loads the alternate language into the language card or into memory. If the alternate language already exists, no action is taken. LOADER.OBJ0 also moves an image of the Monitor ROM from the main logic board to the $F800 location in the language card or in memory.
MASTER	an Applesoft program that runs the MASTER CREATE program, a binary program, to convert an initialized disk into a master disk.
MASTER CREATE	a binary program that creates a master disk that contains a DOS that is placed as high as possible in memory. A master disk is useful on an Apple II computer of any size.
MUFFIN	a binary program to convert 13-sector disks to 16-sector format.
RENUMBER	an Applesoft program that renumbers the lines of a BASIC program or merges two programs.
SLOT#	an Applesoft program that returns the current default values for slot and drive.
START13	an Applesoft program that runs the BOOT13 program.

D.2 Programs on the SAMPLE PROGRAMS Disk

These are the programs on the SAMPLE PROGRAMS disk.

Name	Function
ADDRESS	a demonstration program written in Applesoft that illustrates reading and writing random-text files.
ANIMALS	a game written in Integer BASIC for which you build a data file.
APPLEPROMS	a data file for the RANDOM program. APPLE PROMS contains a parts-list inventory.
APPLEVISION	a demonstration program written in Integer BASIC.
BLACK BOOK	a data file that stores the records of the ADDRESS program.
BRICK OUT	a game program written in Applesoft. Play the game with the arrow keys on the keyboard or with hand controls.
COLOR TEST	a demonstration program written in Applesoft to help you adjust a color TV set.
DELETE.ME.1	a demonstration Applesoft file that lets you practice deleting a file.
DELETE.ME.2	a demonstration Applesoft file that lets you practice deleting a file.
DELETE.ME.3	a demonstration Applesoft file that lets you practice deleting a file.
EXEC DEMO	a demonstration program written in Applesoft that illustrates how an EXEC program is created and runs.
FPBASIC	the Applesoft BASIC language on disk, in binary.
GET TEXT	a demonstration program written in Applesoft that reads text files.
HELLO	an Applesoft greeting program.
INTBASIC	the Integer BASIC language on disk, in binary.
LOADER.OBJ0	a machine-language program loaded by HELLO on the SYSTEM MASTER disk. LOADER.OBJ0 loads the alternate language into the language card or into memory. If the alternate language already exists, no action is taken. LOADER.OBJ0 also moves an image of the Monitor ROM from the main logic board to the $F800 location in the language card or in memory.
LOCK.ME.1	a demonstration Applesoft file that lets you practice locking an unlocked file.
LOCKED.UP.1	a demonstration Applesoft file that lets you practice unlocking a locked file.
LOCKED.UP.2	a demonstration Applesoft file that lets you practice unlocking a locked file.
MAKE TEXT	a demonstration Applesoft program that illustrates creating sequential-text files.
ONERR DEMO	a sample program that illustrates error recovery. It checks to see if a file is locked and, if so, lets you unlock it.
POKER	a demonstration program in Applesoft that illustrates translating machine-language into a text file.
PHONE LIST	a practice program written in Applesoft.
RANDOM	a demonstration program written in Applesoft that illustrates reading and writing with a random-access text file. It uses APPLE PROMS to hold the data.
VERIFY.ME	a demonstration Applesoft file that lets you practice verifying a file.

Downloaded from www.Apple2Online.com

Summary of DOS Operating Concepts and Commands

References to discussions in the *DOS User's Manual* are indicated as Chapter x, DU. References to discussions in this manual are indicated as Chapter x.

E.1 Operating Concepts

DOS operates on Apple II, Apple II Plus, and Apple IIe computers. If information applies only to a specific model, this manual says so explicitly.

E.1.1 Cold Start

See Chapter 1, DU.

The process of turning on the power to your computer (or simulating this same sequence by PR# n) so that DOS is loaded into main memory is

1. Insert a DOS disk into disk drive 1.
2. Turn on your video display screen.
3. Find the power switch on the computer and turn it on.

On an Apple IIe computer, you can simulate a cold start once the power is on. Hold down ⓐ while you press and release CONTROL and RESET; then release ⓐ.

On a standard Apple II or an Apple II Plus, you can simulate a cold start by typing `Sn` CONTROL-P, CONTROL-L, or `Cn00G`, where n is the number of the slot containing the disk controller card.

E.1.2 Warm Start (All Apple II computers)

The process of restarting your computer when the power is already on is

See Chapter 1, DU.

Press (CONTROL) and (RESET) simultaneously and then release them. On some earlier models of Apple II computers, you need to press only (RESET).

A warm start does not reload DOS into main memory, and often does not erase the program in memory.

E.1.3 Initializing a Disk

The process for initializing a disk is

See Chapter 3, DU.

1. Start DOS and put a blank disk in the disk drive.
2. Issue the NEW statement of BASIC to clear the Apple II's memory. Then type in a greeting program, for example

   ```
   10 PRINT "32K DISK INITIALIZED 5 MAY 84"
   20 END
   ```

   To test that the greeting program is correct, type RUN.
3. Assuming you want to name the greeting program HELLO, type INIT HELLO. This formats the disk and writes your program on the disk, giving it the name HELLO.

E.1.4 Creating a Master Disk

This is the procedure for converting an initialized disk into a master disk:

See Chapter 5.

1. First create a greeting program to reflect the master status of the disk by changing an existing greeting program or by typing in a new one. Save the greeting program with a new name (for example, SAVE BIG HELLO). Remove that disk from the drive.
2. Insert the SYSTEM MASTER disk in the drive and type RUN MASTER.
3. Enter the new name of your greeting program. This is the name used in step 1 with the SAVE command. If you want the disk to run some other program—that is, to make a turnkey disk—press (ESC) and enter the name of that program.
4. Replace the SYSTEM MASTER with the initialized disk you want to convert. Press (RETURN) to begin the conversion.

E.1.5 Interpreting Commands

See Chapter 2.

Deferred execution means that a command is issued from a BASIC program. **Immediate execution** means that a command is issued from your keyboard. When a command is preceded by CONTROL-D, only DOS looks at it. To issue a deferred-execution DOS command, use the PRINT command to print a string consisting of CONTROL-D followed by the command. For example

Applesoft only	Integer BASIC and Applesoft

`10 D$=CHR$(4)`
`20 PRINT D$;"CATALOG"`
	`10 PRINT "CATALOG"`
	`20 REM (There's an invisible`
	`30 REM CONTROL-D before`
	`40 REM the C inside first quote mark).`

The CHR$(4) function returns the ASCII equivalent for CONTROL-D.

E.1.6 Capacity

A maximum of 496 disk sectors are available to hold your information. Each disk sector can store up to 256 bytes.

See Appendix B.

An empty text file is 1 sector long; the sector is occupied by the file's empty track/sector list. Empty Applesoft, Integer BASIC, and binary files are 2 sectors long: one sector is the track/sector list, the other is the first program sector, which contains the program's length.

E.1.7 File Types

File types are listed, by code, in a disk's catalog.

E.2 Command Notation

This section describes the DOS command notation used in this manual.

E.2.1 Notation Conventions

UPPERCASE indicates the actual name of a command. Type it exactly as indicated.

lowercase indicates an element that you must supply in a command, for example, the name of a file.

[ ] Square brackets enclose an optional argument in a command. You may include the option or not, as you choose. Do not type the brackets.

E.2.2 Syntax

See Chapter 2.

A file name usually is the first argument following a command word; remaining arguments may appear in any order. The file name must be separated by a comma from any argument that follows, for example

INIT fn [,Vn] [,Sn] [,Dn] indicates the command's syntax.

INIT HELLO, V17, D2 is interpreted this way:

- The command word INIT is uppercase and is typed exactly as shown.
- The argument fn stands for a file name and is replaced by an actual file name in uppercase, HELLO in the example.
- The argument Vn indicates an optional volume number. In the example, it is replaced by V17. (This volume number was chosen arbitrarily.)
- The argument Sn indicates an optional slot number; this example omits it.
- The argument Dn, also optional, indicates that DOS should access the disk in drive 2.

E.2.3 Arguments

See Chapter 2.

fn — in a command, indicates the name of a file that you supply. A file name begins with a letter and may contain from 1 to 30 characters. Any typeable character except the comma may appear in a file name.

See Chapter 2.

n — as an argument of the PR# and IN# commands, indicates a slot number from 1 to 7. As an argument of the MAXFILES command, indicates the number of buffers and may range from 1 to 16. You may use decimal or hexadecimal notation.

See Chapter 2.

n — in all other arguments, indicates a decimal or hexadecimal number. Hexadecimal numbers are preceded by a dollar sign, for example, $FE.

See Chapter 6.

An specifies an address in memory. Replace n with a number from Ø through 65535. With BSAVE, An indicates the address in memory from which the transfer starts. With BLOAD and BRUN, it specifies a target address in memory for loading the binary file; if it is omitted, DOS puts the file in memory starting at the address from which it was saved with the BSAVE command.

Sequential access, see Chapter 3; random access, see Chapter 4.

Bn specifies a byte or character number. Replace n with a number from 0 through 32767; if omitted, DOS defaults to 0. In sequential access, Bn specifies an absolute byte within the file and, for most programs, is not greater than the last byte in the file. In random access, Bn specifies an absolute byte within the record indicated by Rn and, for most programs, is not greater than the last byte in the current record.

See Chapter 2.

Dn specifies the number of a disk drive. Replace n with either 1 or 2. The drive number initially defaults to 1; subsequently it defaults to the last Dn specified (Chapter 2).

See Chapter 6.

Ln specifies length: a number of bytes or characters. Replace n with a number from 0 through 32767. With the OPEN command for random-access files, it specifies the number of bytes that each record holds. With BSAVE, it specifies the number of bytes of memory, starting at the address specified by An. When Ln is omitted, DOS defaults to 1.

POSITION, see Chapter 3; EXEC, see Chapter 5.

Rn with the POSITION and EXEC commands, specifies the number of fields or lines to skip, relative to the current position. Replace n with a number from 0 through 32767. When omitted, DOS defaults to 0, the beginning of the file.

Random-access text files, see Chapter 4.

Rn with READ and WRITE commands for random-access text files specifies a record number. Replace n with a number from 0 through 32767. After OPEN, Rn initially defaults to 0, the first record of the file; after that, it defaults to the last record specified. Rn indicates an absolute record within a random-access file.

Sn — specifies the number of a slot containing the disk controller card of the drive to be accessed. Replace n with a number from 1 to 7. If omitted, DOS uses the default slot number. DOS initially defaults to the number of the slot from which it was started; subsequently, it defaults to the last Sn specified.

See Chapter 2.

Vn — specifies a volume number of a disk. Replace n with a number from 1 through 254. DOS initially defaults to the volume number of the disk from which it was started. After that, DOS defaults to the last Vn specified or read from a disk. When omitted from the INIT command, DOS defaults to 254. When omitted from other DOS commands or when specified as either 0 or just V, DOS ignores the volume specification.

See Chapter 2.

E.3 Command Summary

In this section, the DOS commands are grouped into 5 categories:

1. Nonprogramming commands for accessing disks:

CATALOG	INIT	DELETE	RENAME
LOCK	UNLOCK	VERIFY	RUN
LOAD	SAVE	PR#	IN#

2. Commands that control the programming environment:

FP	INT	MON	NOMON
EXEC	MAXFILES	CHAIN	

3. Programming commands for sequential-access text files:

OPEN	CLOSE	READ	WRITE
APPEND	POSITION		

4. Programming commands for random-access text files:

OPEN	CLOSE	READ	WRITE

5. Commands for binary files:

BRUN	BLOAD	BSAVE

Abbreviated descriptions of all DOS commands are listed below; each command is followed by an example. At the end of each description, you'll find a reference to the chapter where you can find more detailed information.

E.3.1 Nonprogramming Commands for Accessing Disks

CATALOG [,Sn] [,Dn]
Example: CATALOG

displays on the screen the volume number and the names of all the files on your disk. It also displays information about each file: its file type, its size in sectors, and whether it is locked, as indicated by an asterisk (*) beside the file type.

The file types are:

I	Integer BASIC program file, created by SAVE.
A	Applesoft program file, created by SAVE.
T	Text file, created by OPEN and filled by WRITE.
B	Binary memory-image file, created by BSAVE.
R	Relocatable, created by BSAVE.
S	Reserved for future use.

See Chapter 3, DU.

When an individual file exceeds 255 sectors, the catalog display of that file's length starts over at 000. This gives an erroneous impression of remaining space on the disk.

INIT fn [,Sn] [,Dn] [,Vn]
Example: INIT HELLO, V18

See Chapter 3, DU.

organizes the surface of a disk into tracks and sectors and writes zeros in all data fields. INIT also places a copy of DOS and a greeting program on the disk.

DELETE fn [,Sn] [,Dn] [,Vn]
Example: DELETE TEXT

See Chapter 4, DU.

removes an unlocked file from a disk. When the specified file does not exist on the disk, DOS displays the message `FILE NOT FOUND`. To avoid this error, issue an OPEN command before the DELETE.

RENAME fn1,fn2 [,Sn] [,Dn] [,Vn]
Example: RENAME CURRENT, ARCHIVAL, S7, D1

See Chapter 4, DU.

changes the name of a file from fn1 (CURRENT) to fn2 (ARCHIVAL) without affecting the file's contents. If the file was open, it is closed and then renamed. RENAME doesn't check to see whether fn2 already exists.

LOCK fn [,Sn] [,Dn] [,Vn]
Example: LOCK LOVE LETTERS, V31

See Chapter 4, DU.

protects a file from being accidentally destroyed. A locked file is indicated by an asterisk (*) in its disk catalog. You cannot rename, delete, or change a locked file.

UNLOCK fn [,Sn] [,Dn] [,Vn]
Example: UNLOCK DOORS, D2

See Chapter 4, DU.

removes protection from a locked file and makes it possible to rename, delete, and change it.

VERIFY fn [,Sn] [,Dn] [,Vn]
Example: VERIFY FACTS

See Chapter 4, DU.

tests that DOS is able to read the file from a disk into the computer's memory. fn must be unlocked. When DOS can read the file, DOS displays no confirming message. When DOS cannot read the file, you'll see `I/O ERROR`. You can verify any type of file.

RUN fn [,Sn] [,Dn] [,Vn]
Example: RUN ANNUITY, D1

See Chapter 5, DU.

copies a BASIC program, file type A or I, from a disk file into memory and executes it.

LOAD fn [,Sn] [,Dn] [,Vn]
Example: LOAD CARGO, S6, D1

See Chapter 5, DU.

copies a program, file type A or I, from a disk file into memory. Once the program is in memory, you can run it, modify it, or save it as a disk file.

Before loading a new program into memory, DOS closes all text files that are currently open. When a new program is actually loaded, DOS erases any program currently in memory and changes to the BASIC that corresponds to fn's file type.

If the file type is A and Applesoft is not in memory or available from the Applesoft firmware ROM card, DOS tries to load Applesoft from the disk and runs it. If Applesoft is not on that disk, you'll see the message `LANGUAGE NOT AVAILABLE.`

SAVE fn [,Sn] [,Dn] [,Vn]
Example: SAVE COLOR DEMOS, D2

See Chapter 5, DU.

writes the BASIC program currently in main memory to a disk file. When the program is an Applesoft program, DOS saves it as type A. When it is Integer BASIC, DOS saves it as type I.

When fn does not exist, DOS creates a file with that name and stores the program currently in memory in that file. If the disk already contains a file with the same file name and in the same file type, DOS writes over its contents with the current BASIC program. When the disk contains a file with the same file name and a different file type, DOS displays `FILE TYPE MISMATCH`.

PR# n
Example: PR# 6

See Chapter 2.

sends the characters normally printed on the screen to the device connected through slot n. The number sign (#) is part of the command and must be typed.

On the standard Apple II and the Apple II Plus, the command PR# 0 redirects output to the screen. On the Apple IIe, **when you have not turned on the 80-Column Text Card**, use PR#0 to return to a 40-column screen. If the 80-Column Text Card is operating, use PR#3 to return to an 80-column display, or use PR#3 followed by ESC CONTROL-Q to go to a 40-column display.

When a deferred-execution PR# command is not preceded by a CONTROL-D or when no device controller card is in slot n, DOS appears to be disconnected. You have to restart DOS.

IN# n
Example: IN# 2

See Chapter 2.

reads characters from the device connected through slot n (for example, an external terminal) instead of from the keyboard. The number sign (#) is part of the command and must be typed.

To return to the keyboard from some other device, use IN# 0.

When a deferred-execution IN# is not preceded by a CONTROL-D or when no device controller card is in slot n, DOS appears to be disconnected. You have to restart DOS.

E.3.2 Commands That Control the Programming Environment

FP [,Sn] [,Dn]
Example: FP, D2

See Chapter 2.

switches to Applesoft. It resets the pointers in memory so they no longer point to the previous BASIC program and variables; essentially, that program is lost.

If your standard Apple II computer contains the Applesoft firmware card or a Language Card, DOS looks there for Applesoft. When your system does not contain the Applesoft firmware card, DOS tries to load the program FPBASIC from the disk and run it.

INT
Example: INT

See Chapter 2.

switches to Integer BASIC. It resets the pointers in memory so they no longer point to the previous BASIC program and variables; essentially, that program is lost. If Integer BASIC is not present, you'll see `LANGUAGE NOT AVAILABLE`.

MON [C] [,I] [,O]
Examples: MON O and MONICO

See Chapter 2.

displays the action between the computer and the disk. C displays disk commands; I displays the input from the disk to the computer; O displays the output to the disk. You may list the arguments in any order and in any combination, but at least one of them must be present; commas that separate the arguments are optional.

These values for MON remain in effect until you issue a NOMON, FP, or INT command or restart the system; running a program does not cancel them.

NOMON [C] [,I] [,O]
Examples: NOMON C and NOMON I, C

See Chapter 2.

suppresses the action display between the computer and the disk that was turned on by the MON command. C suppresses display of disk commands; I suppresses display of the input from the disk to the computer; O suppresses display of the output to the disk. You may use the arguments in any order and in any combination but at least one of the arguments must be present; commas that separate the arguments are optional.

The command NOMON C, I, O returns the system to its initial state: no commands and information sent between the computer and the disk will be displayed on the screen.

EXEC fn [,Rn] [,Sn] [,Dn] [,Vn]
Example: EXEC UTILITY

See Chapter 5.

directs DOS to take commands from a command file created by a BASIC program rather than from the keyboard. fn indicates the name of the command file, which can contain DOS commands, data, and BASIC statements.

DOS begins executing the command file at the line specified by Rn. When you omit this argument, DOS starts at the first line of the file, which is record 0. If you specify a value of R beyond the end of the file, you'll see an `END OF DATA` message.

Warning

When a program is running under control of a command file, any INPUT statement in the program reads the next field from the command file rather than the keyboard. If the input is an immediate-execution DOS command, the command is executed before the program continues.

MAXFILES n
Example: MAXFILES 6

See Chapter 5.

where n is an integer from 1 to 16, specifies the maximum number of files that can be active at one time. When MAXFILES is executed, DOS reserves a file buffer (595 bytes of memory) for each file. When DOS is started, three file buffers (1785 bytes) are reserved and you can have up to active three files.

All DOS commands except PR#, IN#, and MAXFILES require a file buffer for execution. If you specify MAXFILES 1 and open one file, an attempt to perform most DOS commands displays the message `NO BUFFERS AVAILABLE`.

Warning

Use of MAXFILES moves HIMEM, erasing Integer BASIC programs and Applesoft strings. Use MAXFILES before loading and running a program or as the first line in the program.

CHAIN fn [,Sn] [,Dn] [,Vn]
Example: CHAIN PART TWO, D1, S7

See Chapter 5.

runs a new program and does not close files that are open. It saves in memory the variables from the previous program. Your next program can operate on the results of the previous program and can leave data for subsequent programs. Only Integer BASIC programs can be chained with this command.

E.3.3 Programming Commands for Sequential-Access Text Files

A field is a sequence of characters (1 to 32767 characters) that ends with a carriage return.

A text file (file type T) is a series of fields separated by carriage returns and stored on disk. It can be accessed in two ways: sequential access or random access. Information is stored with the WRITE command and retrieved with the READ command. READ and WRITE need the help of OPEN and CLOSE.

In a sequential-access text file, no length is specified when the file is opened and fields are stored one immediately following the other. That is, DOS writes the first character of each field immediately following the carriage-return character that ended the previous field. Each time the file is opened, DOS starts reading from or writing to the beginning of the file, accessing the fields sequentially.

Warning
All of these sequential-access commands are used in deferred execution. Only CLOSE can be issued in both immediate and deferred execution.

OPEN fn [,Sn] [,Dn] [,Vn]
Example: OPEN SESAME, D2

See Chapter 3.

opens a sequential-access text file so that DOS can read from or write to it. When the specified file does not yet exist, DOS creates it as a text file (type T).

When you open a file, DOS designates a file buffer (595 bytes in memory) for the file and sets the current position for reading and writing to point to the beginning of the file.

CLOSE [fn]
Example: CLOSE CLOSET

See Chapter 3.

tells DOS that you have finished accessing a text file. Before ending, a program must close all open files to ensure that all characters are written and that the file buffers are properly released.

When your program is writing to a file, CLOSE sends all output remaining in the file buffer to the file and then releases the buffer. When CLOSE is used without a file name, DOS closes all open files except a command file.

If a program has an error and stops before it can close all open files, close them by issuing the CLOSE command from the keyboard.

READ fn [,Bn]
Example: READ TLEAVES

See Chapter 3.

tells DOS which sequential-access file to read from and where to start reading. READ is used only after the file is open; fn remains the file from which to read until the next DOS command is specified.

The B option tells DOS to begin reading at the specified byte, relative to the current position.

A field consists of from 1 to 32767 characters, ending with a carriage-return character.

By the Way: An INPUT statement reads characters from the file one field at a time. However, due to the limits of strings and input/output buffers, it is difficult to read fields larger than 255 characters.

WRITE fn [,Bn]
Example: WRITE ADDRESS.DATA

See Chapter 3.

tells DOS which sequential-access text file to write to and where to start writing. WRITE is used only after the file is open; fn remains the file to which to write until the next DOS command is specified.

The B option tells DOS to begin writing at the specified byte, relative to the current position.

By the Way: After this command, PRINT statements send their output to the specified file. Output includes the question-mark prompt character of the INPUT statement if INPUT is used after a WRITE, and error messages if the program does not have an error-handling routine (ONERR GOTO).

APPEND fn [,Sn] [,Dn] [,Vn]
Example: APPEND INFO

See Chapter 3.

opens a sequential-access text file so that your program can write data starting at the end of the file. APPEND opens the file for writing and sets the current position to point to the end of the file.

After this command, the next character written into the file will follow the last sequentially written character presently in the file. APPEND must be followed by a WRITE command that specifies the same file.

POSITION fn [,Rn]
Example: POSITION ADDRESS.DATA, R277

See Chapter 3.

sets the current position for reading or writing. POSITION allows you to skip forward a specified number of fields in the text file before you read or write more information.

When the R option is omitted, the current position is the beginning of the file. When R is specified, the current position is a relative number of fields ahead of the current position.

POSITION scans the contents of the file, character by character, looking for the Rth carriage-return character. It then sets the current position to point to the first byte following that character. Subsequent READ and WRITE commands proceed from that point. If you try to position past the end of the file, you'll see an `END OF DATA` error message.

E.3.4 Programming Commands for Random-Access Text Files

A record is one field or a collection of fields that DOS treats as a unit.

A length argument is specified when a random-access text file is opened. It determines the number of characters in a record.

DOS can start reading from or writing to a specified location in the file. That is, you do not have to read all preceding records in the file and you do not have to read all preceding characters in a given record. DOS accesses subsequent records or a field within a record without regard to what was last accessed.

Warning
All of these random-access commands are used in deferred execution. Only CLOSE can be issued in both immediate and deferred execution.

OPEN fn, Ln [,Sn] [,Dn] [,Vn]
Example: OPEN SESAME, L2

See Chapter 4.

opens a random-access text file. When the specified file does not exist, DOS creates it as a text file (type T). When the file already exists, it must not be open.

The Ln argument, the length of each record, is required. Each time you open the file, specify the same length. DOS uses the length argument to calculate the starting position of each record.

When you open a file, DOS designates a file buffer (595 bytes in memory) for the file, sets the current position to point to the beginning of the file, and sets the record length to the number of bytes specified by the length argument.

CLOSE [fn]
Example: CLOSE ENCOUNTERS

See Chapter 4.

tells DOS that you have finished accessing a text file. Before ending, a program must close all open files to ensure that all characters are written to their files and that the file buffers are properly released.

When your program is writing to a file, the CLOSE command sends all characters remaining in the file buffer to the file and then releases the buffer. When CLOSE is used without a file name, DOS closes all open files except a command file.

When a program contains an error and stops before it can close all open files, issue CLOSE from the keyboard.

READ fn [,Rn] [,Bn]
Example: READ PALMS,R3,B30

See Chapter 4.

tells DOS which random-access text file to read from and where to start reading. READ is used only after the file is open; fn remains the file from which to read until the next DOS command is specified.

The R (record number) argument causes reading to begin at a specified record of the file. When R is omitted, DOS defaults to 0, the first record.

The B (byte) argument causes reading to begin at a specified byte of the record indicated by the R argument. When B is omitted, DOS defaults to 0, the first byte in the record.

By the Way: Subsequent INPUT statements and GET statements of Applesoft read from the specified file rather than the keyboard. The INPUT statement reads characters from the current record, one field at a time.

WRITE fn [,Rn] [,Bn]
Example: WRITE ADDRESS.DATA, R3

See Chapter 4.

tells DOS which random-access file to write to and where to start writing. WRITE is used only after the file is open; fn remains the file to which to write until the next DOS command.

The R (record number) argument causes writing to begin at a specified record of the file. When R is omitted, DOS defaults to 0, the first record.

The B (byte) argument causes writing to begin at a specified byte of the record indicated by the R argument. When B is omitted, DOS defaults to 0, the first byte in the record.

Warning
After the WRITE statement, all output characters that would normally be displayed on the display screen are sent to the file. This includes the question-mark prompt character of the INPUT statement if INPUT is used before WRITE, and error messages if your program doesn't include an error-handling routing (ONERR GOTO).

E.3.5 Commands for Binary Files

BRUN fn [,An] [,Sn] [,Dn] [,Vn]
Example: BRUN SUPER, A$C0A, V75

See Chapter 6.

tranfers a binary file (file type B) stored on the disk to any part of memory and executes the file.

When A is specified, the data is put in memory beginning at the specified address. When the An option is omitted, DOS puts the program in memory starting at the address that was specified in the BSAVE command used to save the file originally.

BLOAD fn [,An] [,Sn] [,Dn] [,Vn]
Examples: BLOAD PICTURE, A8192 (decimal notation)
BLOAD PICTURE, A$2000 (hexadecimal notation)

See Chapter 6.

transfers binary information from a disk file to any part of memory.

When the An option is specified, the data is placed in memory beginning at the specified address. When A is omitted, DOS loads it starting at the address that was specified in the BSAVE command used to save the file originally.

Warning
When An is specified, a machine-language program may no longer be executable at its new address.

BSAVE fn , An , Ln [,Sn] [,Dn] [,Vn]
Examples: BSAVE PICTURE, A16384, L8192 (decimal notation)
BSAVE PICTURE, A$4000, L$2000 (hexadecimal notation)

See Chapter 6.

transfers binary data from memory to a file on the disk. When the file does not exist, DOS creates it as a binary file (type B). When it does exist, DOS overwrites its contents with the specified contents of memory. The An (address) and Ln (length) arguments allow transfer from any part of memory.

Glossary

The glossary includes terms that appear in this manual as well as some you may encounter in other Apple publications. The first time an important term is used in this manual, it is shown in boldface type so you will know that the word is defined here.

address A number used to identify a location in the computer's memory.

Apple IIe A personal computer in the Apple II family.

Apple IIe 80-Column Text Card A peripheral card that plugs into the Apple IIe's auxiliary slot and converts the display of text from 40- to 80-column width.

Apple IIe Extended 80-Column Text Card A peripheral card that plugs into the Apple IIe's auxiliary slot and converts the text display from 40- to 80-column width. It also extends the Apple II's memory capacity by 64K bytes.

Applesoft An extended version of the BASIC programming language used with Apple II computers and capable of processing numbers in floating-point. An interpreter for executing Applesoft programs is built into firmware in the Apple IIe and Apple II Plus.

application program A program that puts the resources and capabilities of the computer to use for some specific purpose or task, such as word processing, data-base management, graphics, or telecommunications.

argument The value on which a function operates.

array A collection of variables referred to by the same name and distinguished by means of numerical subscripts.

ASCII American Standard Code for Information Interchange; an information code in which the numbers from 0 to 127 represent alphanumeric or control characters. ASCII code is used for representing text inside a computer and for transmitting text between computers or between a computer and a peripheral device.

assembler A language translator that converts a program written in assembly language into an equivalent program in machine language.

assembly language A low-level programming language in which individual machine-language instructions are written in a symbolic form more easily read by a person than machine language.

auxiliary slot The special expansion slot inside the Apple IIe used for the 80-Column or Extended 80-Column Text Card.

BASIC Beginner's All-Purpose Symbolic Instruction Code; a high-level programming language designed to be easy to learn and use. Two versions of BASIC are available for use with Apple II computers: Applesoft (built into the Apple IIe and the Apple II Plus in firmware) and Integer BASIC (provided on the SYSTEM MASTER disk).

binary A numeric representation in terms of the powers of 2, using the digits 0 and 1. Commonly used in computers, since the values 0 and 1 can easily represent such states as the presence or absence of electric current, positive or negative voltage, and a white or black dot on the screen.

binary file A file containing information not expressed in terms of 0 and 1.

bit A binary digit (0 or 1); the smallest possible unit of information, consisting of a simple two-way choice, such as yes or no, on or off, positive or negative, something or nothing.

boot To start a computer by loading a program into memory from an external storage medium such as a disk. Often accomplished by first loading a small program whose purpose is to read the larger program into memory. The program is said to "pull itself up by its own bootstraps"; hence the term "booting."

boot disk A disk containing software recorded in the proper form to be loaded into the Apple II's memory in order to set the system into operation. Also called a startup disk.

buffer An area of the computer's memory reserved for a specific purpose, such as to hold graphical information to be displayed on the screen or text characters being read from the keyboard or some peripheral device. Often used as an intermediary "holding area" for transferring information between devices operating at different speeds, such as the computer's processor and a printer or disk drive. Information can be stored into the buffer by one device and then read by the other at a different speed.

bug An error in a program that causes it not to work as intended.

byte A unit of information consisting of a fixed number of bits; on the Apple II, one byte consists of eight bits and can hold a value from 0 to 255.

call To request the execution of a subroutine or function.

character A letter, digit, punctuation mark, or other symbol used in printing or displaying information that can be read by people; see control character.

chip The small piece of semiconducting material (usually silicon) on which an integrated circuit is fabricated. The word chip properly refers only to the piece of silicon itself and is often used for an integrated circuit and its package; see integrated circuit.

code (1) A number or symbol used to represent a discrete piece of information in a compact form. (2) Statements or instructions comprising a program.

cold start The process of starting up the Apple II when the power is first turned on by loading the operating system into main memory, then loading and running a program. Certain commands and control characters simulate a cold start.

command A communication from the user to a computer system directing it to perform some immediate action. A command can by typed from the keyboard or embedded in a program.

command file A program that runs other programs. The EXEC command runs a command file, which can contain both DOS commands and BASIC statements.

component A part; in particular, a part of a computer system.

connector The physical joining place for electrical or electronic devices. Also a device, such as a plug, socket, or jack, used to connect one hardware component of a system to another.

control character A character that controls or modifies the way information is printed or displayed. Control characters have ASCII codes between 0 and 31 and are typed from the keyboard by holding down CONTROL while typing some other character. For example, pressing CONTROL-M means "return to the beginning of the line" and is equivalent to RETURN.

controller card A card that connects a peripheral device such as a printer or disk drive to the Apple II and controls the operation of the device.

cursor A marker or symbol displayed on the screen that marks where the user's next action will take effect or where the next character typed from the keyboard will appear.

debug To locate and correct an error or the cause of a problem or malfunction in a computer system. Typically used to refer to software-related problems.

decimal The common form of number representation in which numbers are expressed in terms of the powers of 10, using the digits 0 to 9.

default A value, action, or setting that is automatically used by a computer system when no other explicit information is given. For example, if a command to run a program from a disk does not identify which disk drive to use, DOS automatically uses the same drive that was used in the last operation.

deferred execution The saving of an instruction in a program for execution at a later time as part of a complete program; occurs when a statement is typed with a line number.

digit (1) One of the characters 0 to 9, used to express numbers in decimal form. (2) One of the characters used to express numbers in some other form, such as 0 and 1 in binary or 0 to 9 and A to F in hexadecimal.

dimension the maximum size of one of the subscripts of an array.

directory A list of all files stored on a disk; also called a catalog.

disk An information storage medium consisting of a circular magnetic surface on which information can be recorded in small magnetized spots, similarly to the way sounds are stored on tape.

disk controller card A peripheral card that connects one or two disk drives to the Apple II and controls their operation.

disk drive A peripheral device that writes and reads information on the surface of a magnetic disk.

diskette A term sometimes used for the small (5¼-inch) flexible disks.

element A member of a set or collection; specifically, one of the individual variables making up an array.

EXEC file A program that runs other programs when it is executed by the EXEC command.

expansion slot A connector inside the Apple II computer in which a peripheral card can be installed; also called peripheral slot.

field A sequence of from 1 to 32767 characters that ends with a carriage-return character.

fixed-point A method of representing numbers in the computer in which the decimal point (more correctly, the binary point) is considered to occur at a fixed position in the number. Typically, the point is considered to lie at the right end of the number, so that the number is interpreted as an integer. Fixed-point numbers of a given length represent a narrower range than floating-point numbers of the same length, but with greater precision.

floating-point A method of representing numbers in the computer in which the decimal point (more correctly, the binary point) is permitted to "float" to different positions within the number. Some of the bits within the number itself are used to keep track of the point's position. Floating-point numbers of a given length represent a wider range than fixed-point numbers of the same length, but with less precision.

format (1) The form in which information is organized or presented. (2) To specify or control the format of information. (3) To prepare a blank disk to receive information by dividing its surface into tracks and sectors; also initialize.

function A preprogrammed calculation that can be carried out on request from any point in a program.

hardware Those components of a computer system consisting of physical (electronic or mechanical) devices.

hexadecimal The representation of numbers in terms of powers of 16, using the digits 0 to 9 and A to F. Hexadecimal numbers are easier for people to read and understand than binary numbers and can be converted easily and directly to binary: a hexadecimal digit corresponds to a sequence of four binary digits, or bits.

high-order byte The more significant half of a memory address or other two-byte quantity. In the Apple II's 6502 microprocessor, the low-order byte of an address is usually stored first and the high-order byte second.

IC See integrated circuit.

immediate execution The execution of an instruction of a program as soon as it is typed; occurs when the line is typed without a line number.

indexed addressing A method of specifying memory addresses used in machine-language programming.

initialize (1) To set to an initial state or value in preparation for some computation. (2) To prepare a blank disk to receive information by dividing its surface into tracks and sectors; also format.

initialized disk a disk that is organized into tracks and sectors and contains DOS and a greeting program. This disk will contain a DOS that is placed in the same memory location that it had in the computer on which the disk was prepared. The INIT command is used to create an initialized disk.

input (1) Information transferred into a computer from some external source, such as the keyboard, a disk drive, or a modem. (2) The act or process of transferring such information.

instruction A unit of a machine-language or assembly-language program corresponding to a single action for the computer's processor to perform.

integer A whole number; represented inside the computer in fixed-point form.

Integer BASIC A version of the BASIC programming language used with the Apple II computers. Integer BASIC is older than Applesoft and capable of processing numbers in integer (fixed-point) form only. The SYSTEM MASTER disk includes an interpreter for creating and executing programs in Integer BASIC and automatically loads it into memory when the computer is started with that disk.

integrated circuit An electronic component consisting of many circuit elements fabricated on a single piece of semiconducting material, such as silicon; see chip.

interface card A peripheral card that implements a particular interface by which the computer can communicate with a peripheral device such as a printer or modem.

interrupt A temporary suspension in the execution of a program by a computer in order to perform some other task, typically in response to a signal from a peripheral device or other source external to the computer.

I/O Input/output; the transfer of information into and out of a computer. See input, output.

I/O link A fixed location that contains the address of an input/output subroutine in the Apple II Monitor program.

K Two to the tenth power, or 1024 (from the Greek root kilo, meaning one thousand); for example, 64K equals 64x1024, or 65,536.

keyboard The set of keys connected to the computer, similar to a typewriter keyboard, for entering information into the computer.

kilobyte A unit of information consisting of 1K (1024) bytes, or 8K (8192) bits; see K.

KSW The symbolic name of the location in the Apple II's memory where the standard input link is stored; stands for "keyboard switch." See I/O link.

load To transfer information from a peripheral storage medium such as a disk into main memory.

loader A program that brings BASIC or a binary file from disk into main memory.

low-order byte The less significant half of a memory address or other two-byte quantity. In the Apple II's 6502 microprocessor, the low-order byte of an address is usually stored first and the high-order byte second.

machine language The form in which instructions to a computer are stored in memory for direct execution by the computer's processor. Each model of computer processor has its own form of machine language.

main memory The memory component of a computer system that is built into the computer itself and whose contents are directly accessible to the processor.

master disk a disk containing a DOS that is placed as high as possible in memory. It can operate on any system 16K or larger. The MASTER program creates a master disk from an initialized disk.

memory A hardware component of a computer system that can store information for later retrieval; see main memory, random-access memory, read-only memory, read-write memory.

memory location A unit of main memory that is identified by an address and can hold a single item of information of a fixed size; in the Apple II, a memory location holds one byte of information.

microprocessor A computer processor contained in a single integrated circuit.

Monitor program A system program built into Apple II firmware, used for directly inspecting or changing the contents of memory and for operating the computer at the machine-language level.

output (1) Information transferred from a computer to an external destination, such as display screen, disk drive, printer, or modem. (2) The process of transferring such information.

Pascal A high-level programming language. A version called Apple II Pascal is used with the Apple II family of computers.

peripheral card A removable printed-circuit board that plugs into one of the Apple II's expansion slots and expands or modifies the computer's capabilities by connecting a peripheral device or performing some subsidiary or peripheral function.

peripheral slot See expansion slot.

pointer An item of information consisting of the memory address of some other item. For example, Applesoft maintains internal pointers to (among other things) the most recently stored variable, the most recently typed program line, and the most recently read data item.

processor The hardware component of a computer that performs the actual computation by directly executing instructions represented in machine language and stored in main memory.

program A set of instructions describing actions for a computer to perform in order to accomplish some task, conforming to the rules and conventions of a particular programming language.

prompt To remind or signal the user that some action is expected, typically by displaying a distinctive symbol, a reminder message, or a menu of choices on the display screen.

prompt character A text character displayed on the screen to prompt the user for some action. Often also identifies the program or component of the system that is doing the prompting; for example, the prompt character] is used by the Applesoft interpreter, > by Integer BASIC, and * by the system Monitor program.

radio-frequency modulator A device for converting the video signals produced by a computer to a form that can be accepted by a television receiver.

RAM See random-access memory.

random-access memory Memory in which the contents of individual locations can be referenced in an arbitrary or random order. This term is often used incorrectly to refer to read-write memory, but strictly speaking, both read-only and read-write memory can be accessed in random order.

read To transfer information into the computer's memory from a source external to the computer (such as a disk drive or modem) or into the computer's processor from a source external to the processor (such as the keyboard or main memory).

read-only memory Memory whose contents can only be read; used for storing firmware. Information is written to read-only memory once, during manufacture; it then remains there permanently, even when the computer's power is turned off; it can never be erased or changed.

read-write memory Memory whose contents can be both read and written; often misleadingly called random-access memory, or RAM. The information contained in read-write memory is erased when the computer's power is turned off and is permanently lost unless it has been saved on a more permanent storage medium, such as a disk.

register A location in a computer processor where information is held and modified under program control.

sector A portion of the recording surface of a disk consisting of a fixed fraction of a track. Under DOS 3.3, there are 16 sectors per track.

software Those components of a computer system consisting of programs that determine or control the behavior of the computer.

startup disk A disk containing software recorded in the proper form to be loaded into the Apple II's memory in order to set the system into operation. Sometimes called a boot disk.

statement A unit of a program in a high-level language specifying an action for the computer to perform, typically corresponding to several instructions of machine language.

subroutine A part of a program that can be executed on request from any point in the program, and which returns control to the point of the request on completion.

subscript An index number used to identify a particular element of an array.

syntax The rules governing the structure of statements or instructions in a programming language.

text (1) Information presented in the form of characters readable by humans. (2) The display of characters on the Apple II's display screen.

text file A file containing information expressed in text form.

text window An area on the Apple II's display screen within which text is displayed and scrolled.

track A portion of the recording surface of a disk consisting of a single circular band at a fixed distance from the center of the disk. Under DOS 3.3, there are 35 tracks on a disk.

turnkey disk A disk that executes a specific program when you start it up.

variable (1) A location in the computer's memory where a value can be stored. (2) The symbol used in a program to represent such a location.

video monitor A display device that receives video signals by direct connection only, and which cannot receive broadcast signals such as commercial television. Can be connected directly to the Apple II computer as a display device.

warm start The process of restarting the Apple II after the power is already on, without reloading the operating system into main memory and often without losing the program or information already in main memory.

wildcard character The equal sign (=), which may represent the files within a directory or those files that share a file pattern.

write To transfer information from the computer to a destination external to the computer, such as a disk drive, printer, or modem, or from the computer's processor to a destination external to the processor, such as main memory.

write-enable notch A square cutout in one edge of a disk's jacket that permits information to be written on the disk. If there is no write-enable notch, or if it is covered, information can be read from the disk but not written onto it.

write-protect To protect the information on a disk by covering the write-enable notch with a write-protect tab, preventing any new information from being written onto the disk.

write-protect tab A small adhesive sticker used to write-protect a disk by covering the write-enable notch.

Downloaded from www.Apple2Online.com

1.3.5 The UNLOCK Command

The UNLOCK command unlocks a file, removing the file protection so that you can delete, rename, or change the file.

UNLOCK fn [,Sn] [,Dn] [,Vn]

fn	is the name of the file to unlock.
[,Sn]	where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.
[,Dn]	where n is 1 or 2, specifies the drive that holds the disk containing fn. If omitted, DOS uses the default drive number.
[,Vn]	where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS ignores the volume specification.

1.3.6 The DELETE Command

The DELETE command lets you remove a file from a disk.

DELETE fn [,Sn] [,Dn] [,Vn]

fn	is the name of the file to be deleted; fn must be unlocked. If the file does not exist, you'll see `FILE NOT FOUND`.
[,Sn]	where n is a number from 1 to 7, specifies the slot containing the disk controller card of the drive to be accessed. If omitted, DOS uses the default slot number.
[,Dn]	where n is 1 or 2, specifies the drive of the disk containing fn. If omitted, DOS uses the default drive number.
[,Vn]	where n is a number from 0 to 254, specifies the volume number of the disk to be accessed. If you specify 0, just V, or omit the option, DOS ignores the volume specification.

Index

CONTROL-RESET 181
← 33
→ 25, 33
= 141, 143
] 15, 21
> 15, 21
; 26, 27, 50
, 27, 29, 57, 173
35
& 38
$ 7
* 7, 24, 124, 173, 187
> 63999 41
*** SYNTAX ERROR 167
#3EAG 127
?SYNTAX ERROR 167
13-sector disks 139-145
16K RAM Card 20
3D0G 24, 29
*3D0G 124
3D3G 124
3D3G 29
3EAG 124
6502 Assembler/DOS Tool Kit 149
80-Column Text Card 33, 189

A

A register 160, 161
ADDRESS program 79, 178
addresses, binary 118
addressing tracks and sectors 159-163
ampersand 38
An argument 118, 185
  with BLOAD 121, 197
  with BRUN 120, 197
  with BSAVE 122, 197
ANIMALS program 178
APPEND command 28, 69, 175, 194
Apple II Reference Manual 24
Apple Language System 109
Apple Writer 2.0 91
APPLEPROMS program 178
Applesoft BASIC 13, 20, 23, 25, 169, 190
  and CHAIN 103
  and MAXFILES 102
Applesoft BASIC Programmer's Reference Manual 3, 20, 28, 62
Applesoft Card 20
Applesoft II BASIC Programming Manual 132
APPLESOFT language file 110
APPLESOFT program 21, 169, 177
Applesoft Tutorial 3, 19
Applesoft/DOS Tool Kit 91
APPLEVISION program 178
arguments 184-186
array(s)
  definition 56
  with CHAIN 103
ASCII 25
asterisk 7, 10, 24, 124, 173, 187
AWAY program 94

B

BASIC 20
BASIC program files 20-24
BASICS disk 144, 145
binary addresses 118
binary files 117-123, 158
  commands 119-123, 197
BLACK.BOOK 79, 178
BLOAD command 119, 121, 134, 197
Bn argument 171, 185
  with READ 68, 193, 196
  with WRITE 68, 196
BOOT13 program 144, 177
BRICK OUT program 178
BRUN command 119, 120, 197
BSAVE command 119, 120, 122, 158, 197

buffer, file 65, 85, 101, 191, 195
buffering 101
bugs 28
byte(s) 101
  link 149

C

C argument, with MON 29, 190
C argument, with NOMON 30, 190
C00G 161
CALL -151 123, 124
CALL -868 30
CALL 1002 34, 127
CALL 3072 161
CALL statement 129
CAPS LOCK 7
CAPTURE routine 97
capturing lines 96
carriage return 27, 28, 47, 192
CATALOG command 7, 151, 187
CHAIN command 91, 192
  and Applesoft 103
  and Integer BASIC 103
CHAIN program 103, 105, 177
CHR$ function 25
CLOSE command 48, 100, 174
  with random access 85, 195
  with sequential access 66, 193
Cn00G 181
cold start 181
COLOR TEST program 178
comma 27, 29, 38, 57, 173
  in fields 59
command
  arguments 184-186
  defaults 6
  description 5
  notation 5, 183
  program environment 190-192
  summary 186-197
  syntax 184
command file(s) 20, 91-99
  and INPUT 100
  creating 94
commands
  binary 119-123, 197
  random-access 84-88, 195-196
  nonprogramming 187
  sequential-access 65-71, 192-194
contents of sectors 148
control characters 25, 33
CONTROL-A 59
CONTROL-B 24
CONTROL-C 100, 124, 143
CONTROL-D 25, 26, 28, 34, 91, 92, 126, 127, 183, 189
CONTROL-H 33
CONTROL-K 24
CONTROL-L 181
CONTROL-M 33
CONTROL-P 124, 127, 181
CONTROL-RESET 182
CONTROL-S 64
CONTROL-U 33
CONVERT13 program 140, 177
converting 13-sector disks 140-143
COPY program 170, 177
COPY.OBJ0 program 177
COPYA program 177
COUT1 routine 128
CSWH 127
CSWL 127
current position pointer 50, 69

D

D$ variable 26, 28
debugging 28-30
defaults 6
deferred execution 20, 28, 183
DELETE command 11, 187
DELETE.ME.1 program 178
DELETE.ME.2 program 178
DELETE.ME.3 program 178
deleting files 66, 158, 187
device-characteristics table 160, 161, 163
direct DOS commands 7-15
directory, disk 7, 148, 151-155
DISK FULL 173
disk(s)
  13-sector 139-145
  access commands 187
  capacity 183
  initializing 8, 101, 187
  master startup 108, 182
  protection 10
  startup 108
  turnkey 108, 182
Dn argument 6, 173, 185
DO'ER command file 92
DOIT command file 94
dollar sign 7
DOS error messages 167-175
DOS
  and Monitor 127
  defaults 6
  disks 3
  entry points 133
  environment 3, 4
  reconnect routine 133
  requirements 4

syntax 5
vectors 163-164
DOS User's Manual 3
drive number option 6
DUPLICATE LINE NUMBERS 41

E

EDASM 91
En command 37
END command 37
END OF DATA 69, 171, 191, 194
equal sign 141, 143
error messages, DOS 167-175
error trapping 30-33
ESC 33
ESC CONTROL-O 33, 35, 189
EXEC command 20, 39, 52, 91, 99, 191
EXEC DEMO program 92, 178
EXEC file 20
EXTRA IGNORED 54

F

FID program 177
field(s)
definition 47
random access 47
sequential access 47
with commas 59
file buffer 85, 101, 147, 191, 195
definition 65
FILE LOCKED 10, 31, 173
file name 184
FILE NOT FOUND 9, 11, 12, 13, 172, 187
file protection 10, 11
file type 13, 15, 47, 149, 154
FILE TYPE MISMATCH 13, 174, 189
file types 174, 183, 187
file(s)
deleting 66, 158, 187
locking 7, 187, 188
protecting 188
renaming 187
unlocking 188
files
binary 117-123, 158
command 91-99
random-access 47, 75-88, 151
sequential-access 47-71
text 48, 192
FILEM program 177
FIRST command 37
floating point 20, 23
fn argument 5, 184
Fn command 37
FP command 21, 22, 23, 29, 100, 169, 175, 190
FPBASIC language file 110
FPBASIC program 169, 177, 178, 190
functions 25
with CHAIN 103

G

GET command 171
GET statement 54, 59, 86, 196
GET TEXT program 63, 91, 92, 178
greater-than-sign 15, 21

H

H command 38
HELLO program 177, 178
Applesoft 26
Integer BASIC 26
hexadecimal 6, 7, 117
hidden characters 33
high-resolution graphics 132
HIMEM 129, 133, 175
and MAXFILES 102, 132, 191
values 132
hold buffer 39
HOLD command 38
HOLD FILE IN USE 41

I

I argument, with MON 29, 190
I argument, with NOMON 30, 190
I/O block 160, 161
format 162
I/O ERROR 12, 15, 139, 172, 188
immediate execution 19, 183
In command 37
IN# 0 36
IN# command 34, 36, 126, 127, 170, 189, 191
INC command 37
INIT command 182, 187
initializing disks 8, 101, 182, 187
INPUT command 171
input registers, DOS 126
input registers, Monitor 125
INPUT statement 54, 86, 191, 196
with command files 100
INT command 21, 22, 23, 29, 100, 169, 175, 190
INTBASIC language file 110
INTBASIC program 177, 178
Integer BASIC 4, 13, 20, 21, 23, 169, 190
and CHAIN 103
and MAXFILES 102

Integer Card 20
intercepting errors 30-33

J

K

KEYIN routine 128
KSWH 127
KSWL 127

L

Language Card 6, 20, 109, 169, 190
LANGUAGE NOT AVAILABLE 13, 21, 109, 169, 188, 190
left-arrow key 33
LIMITED MEMORY... 41
LINE INCREMENT 41
LINE INCREMENT TOO LARGE 41
line numbers 37
LINE TOO LONG 41
line-number references 39
link bytes 149
link pair 149
LIST command 94
Ln argument 118, 185
with OPEN 84
with BSAVE 122, 197
with OPEN 171, 195
with READ 171
with WRITE 171
LOAD command 14, 21, 169, 188
LOADER.OBJ0 program 109, 110, 169, 177, 178
LOADING APPLESOFT... 21
LOADING INTEGER... 21
LOCK command 10, 178
LOCK.ME.1 program 178
locked files 7, 187, 188
LOCKED.UP.1 program 178
LOCKED.UP.2 program 178
LOMEM 133, 175

M

M command 38
machine language 98
MAKE TEXT program 61, 70, 96, 178
MAKE.DOIT program 94
maps, memory 129-133
MASTER CREATE program 106, 177
master disk 182
MASTER program 91, 106, 110, 177
master startup disk 108
MAXFILES command 39, 65, 91, 101, 174, 175, 191
and Applesoft 102
and HIMEM 132
and Integer BASIC 102
memory address 118
memory maps 129-133
memory pages 132
memory, and high-resolution graphics 132
MERGE command 38
MON command 28, 29, 190
with GET 59
Monitor and DOS 127
Monitor I/O registers 125
monitor input link 127
monitor output link 127
Monitor program 24, 124, 127
prompt character 124
MUFFIN program 140, 177

N

n 5, 7
n argument 184
n(CONTROL)-(K) command 127
n(CONTROL)-(P) command 127
NEW 26, 100
NO BUFFERS AVAILABLE 102, 174, 191
NO LINES IN RANGE 41
NO PROGRAM IN MEMORY 38, 42
NOMON command 28, 29, 30, 190
with GET 59
NOT DIRECT COMMAND 175
notation, command 5, 183
NOTRACE statement 28
number sign (#) 35
numbers 6

O

O argument, with MON 29, 190
O argument, with NOMON 30, 190
ONERR DEMO program 31, 32, 178
ONERR GOTO statement 30, 31, 32
OPEN command 28, 48, 87, 94, 175, 187
with random access 84, 195
with sequential access 65, 192
OUT OF MEMORY 42
output registers, DOS 126
output registers, Monitor 125

P

Page 3 163-164
pages 132
PEEK function 30, 31, 98
peripheral device 34
PHONE LIST program 178
pointer, current position 50, 69
POKE function 98, 127, 175
POKER program 98, 178
POSITION command 28, 68, 87, 171, 175
PR# command 34, 35, 126, 127, 170, 174, 189, 191
PR#0 33
PR#3 33, 35
PRINT command 94
PRINT to a file 53
PROGRAM ON HOLD... 38
PROGRAM TOO LARGE 175
prompt character(s) 15, 21
  Monitor 24, 124
protecting files 188

R

RAM 4, 20
RANDOM program 77, 178
random vs. sequential access 49
random-access commands 84-88, 195-196
random-access files 47, 75-88, 151
random-access memory 4, 20
RANGE ERROR 101, 170
READ command 28, 175
  with random access 77, 87, 196
  with sequential access 68, 193
read-only memory 20
reconnecting DOS 133
record length 76, 84, 85
records, 47, 75
references, line number 39
registers, I/O 125-127
REMARK statement 39
RENAME command 9, 31, 187
renaming files 187
RENUMBER program 36-42, 177
  commands 38
  error messages 41
(RESET) 39, 182
resident language 20
(RETURN) 7
right bracket 15, 21
right-arrow key 25, 33
Rn argument 185
  with EXEC 99, 171, 185, 191
  with POSITION 69, 171, 185, 194
  with READ 87, 171, 185, 196
  with WRITE 86, 185, 196
ROM 20
RUN command 13, 14, 21, 169, 188
RWTS subroutine 159-163

S

SAMPLE PROGRAMS disk 4, 32, 61, 63, 77, 79, 91, 92, 98, 178
SAVE command 15, 158, 188
sector(s) 147-149
  addressing 159-163
  allocation 158
  contents 148
  definition 147
  unassigned 151
semicolon 26, 27, 50
sequential vs. random access 49
sequential-access commands 65-71, 192-194
sequential-access files 47-71
slot 0 6
slot number 6
slot number option 6
SLOT# program 177
Sn argument 6, 173, 181, 186
  with RENUMBER 37
START command 37
START13 program 144, 145, 177
starting a program 34
startup disk 108
storage 147-165
summary card 3
summary, DOS commands 186-197
summary, DOS operating concepts 181-186
switching languages 22
SYNTAX 42
SYNTAX ERROR 167, 173
syntax, command 5, 184
SYSTEM MASTER disk 3, 36, 103, 105, 106, 109, 110, 140, 144, 145, 169, 177, 182

T

(TAB) 27
text files 48, 192
TEXT statement 62
text window 62
TRACE 28, 87
track bit map 157
tracks 147-149, 158
  addressing 159-163
  allocation 158
  definition 147
track/sector allocation 158

track/sector list 148, 149-151, 183
track/sector pairs 150, 151
translating machine language 98
turnkey disk 108, 182

U

UNABLE TO READ 139
UNABLE TO WRITE 139
UNLOCK command 11, 188
unlocking files 188

V

VERIFY command 12, 173, 188
VERIFY.ME program 178
Vn argument 7, 186
VOLUME MISMATCH 172
volume number 8
volume number option 7
volume table of contents 148, 152, 155-156
VTOC 155, 157, 158

W

warm start 182
wildcard 141, 143
window, text 62
WRITE command 28, 69, 94, 158, 175
  with random access 76, 86, 196
  with sequential access 67, 193
WRITE PROTECTED 170
write-protection 10

X

Y

Y register 160, 161

Z

zero page 165

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

030-0536-A