NOTICE
Apple Computer Inc. reserves the right to make improvements in the product described in this manual at any time and without notice.

DISCLAIMER OF ALL WARRANTIES AND LIABILITY
APPLE COMPUTER INC. MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL OR WITH RESPECT TO THE SOFTWARE DESCRIBED IN THIS MANUAL. ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. APPLE COMPUTER INC. SOFTWARE IS SOLD OR LICENSED "AS IS." THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE IS WITH THE BUYER. SHOULD THE PROGRAMS PROVE DEFECTIVE FOLLOWING THEIR PURCHASE, THE BUYER (AND NOT APPLE COMPUTER INC., ITS DISTRIBUTOR, OR ITS RETAILER) ASSUMES THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR, OR CORRECTION AND ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES. IN NO EVENT WILL APPLE COMPUTER INC. BE LIABLE FOR DIRECT, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE SOFTWARE, EVEN IF APPLE COMPUTER INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES OR LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

This manual is copyrighted. All rights are reserved. This document may not, in whole or part, be copied, photocopied, reproduced, translated or reduced to any electronic medium or machine readable form without prior consent, in writing, from Apple Computer Inc.

© 1980, 1981 by APPLE COMPUTER INC.
10260 Bandley Drive
Cupertino, California 95014
(408) 996-1010

The word Apple and the Apple Logo are registered trademarks of APPLE COMPUTER INC.

APPLE Product #A2L0036
(030-0115-B)

WARNING: This equipment has been certified to comply with the limits for a Class B computing device, pursuant to Subpart J of Part 15 of FCC Rules. Only peripherals (computer input/output devices, terminals, printers, etc.) certified to comply with the Class B limits may be attached to this computer. Operation with non-certified peripherals is likely to result in interference to radio and TV reception.
# TABLE OF CONTENTS

## PREFACE

## CHAPTER 1

### INSTALLATION AND HANDLING

1. Unpacking
2. Connecting the Cable
3. Installing the Controller
4. Installing Multiple Disk Drives
5. Care of the Disk II and Diskettes
6. Inserting and Removing Diskettes

## CHAPTER 2

### GETTING STARTED

1. Background
2. Special Keys
3. Booting DOS
4. If Booting Doesn't Work
5. Initializing New Diskettes
6. LOADING and SAVING with DOS
7. CATALOG
8. What's in a Name?
9. RENAMEing Files
10. DELETEing Files
11. Recovering from Accidental Resets

## CHAPTER 3

### EXERCISING OPTIONS

12. Drive, Slot, and Volume Options
13. Syntax
14. INIT
15. LOAD, RUN, and SAVE
16. DELETE
17. A Scenario: boot, CATALOG, SAVE, RUN, and DELETE
18. Moving Between Languages: LP and IPL
19. Use of DOS From Within a Program

## CHAPTER 4

### PLAYING SAFE

34. Creating a Turnkey System
35. LOCK and UNLOCK
36. VERIFY
37. Write-Protecting a Disk
38. Protecting Yourself against Disaster
39. Using the COPY Program

## CHAPTER 5

### MORE "HOUSEKEEPING" INFORMATION

42. Debugging: MODN and NCOM
43. MAXFILES
44. TRAGE
45. Using the MASTER UPDATE Program

## CHAPTER 6

### USING SEQUENTIAL FILES

48. Text Files: An Introduction
49. Sequential Files: Some Examples
50. OPENing and CLOSING Sequential Files
51. WRITING Sequential Files
52. READing Sequential Files
53. More on Sequential Files: APPEND and POSITION
54. Byte-ing Off More

## CHAPTER 7

### AUTO APPLE

74. Controlling the Apple via a Text File: EXEC
75. Creating an EXEC File
76. Capturing Programs in a Text File
77. Converting Machine-Language Routines to BASIC
78. MAXFILES and Integer BASIC Programs
79. EXECutive Session
# Chapter 8

**Using Random-Access Files**

82 Random-Access Files: How They Work  
82 A Specific Record  
84 Multiple Records  
86 A Demonstration: The RANDOM Program  
88 WRITing and READING Random-Access Files

---

# Chapter 9

**Using Machine Language Files**

92 Machine Language Files  
92 BSAVE  
93 BLOAD  
94 BRUN  
94 The RTS Subroutine

---

# Chapter 10

**Input, Output, and Chaining**

100 Selecting I/O Devices: IN$, PR$, and Other Stuff  
106 Integer BASIC CHAIN  
106 Applacsoft Chain

---

# Appendix A

**File Types Used with DOS Commands**

110 By DOS Command  
111 By File Type

---

# Appendix B

**DOS Messages**

114 ONERR GOTO Codes  
115 Discussion

---

# Appendix C

**Format of Diskette Information**

124 Overview of the Storage Process  
124 WRITing into a Sequential Text File  
126 WRITing into a Random-Access Text File  
126 How DOS WRITES into Text Files: General Procedure  
127 Contents of File Sectors  
129 The Track/Sector List  
129 The Diskette Directory  
132 Volume Table of Contents  
133 Track Bit Map  
135 Track and Sector Allocation Order  
136 Retrieving Information from the Disk  
136 READING from a Sequential File  
137 READING from a Random-Access File

---

# Appendix D

**Memory Usage**

148 Memory Areas Over-Written When Booting DOS  
148 Memory Areas Used by DOS and Either BASIC  
148 HIMEM Set by Booting DOS

---

# Appendix E

**Entry Points and Schematics**

144 DOS Entry Points  
145 Circuit Schematic: Disk II Interface  
146 Circuit Schematic: Disk II Analog Board

---

# Appendix F

**Summary of DOS Commands**

148 Notation  
151 File Names  
151 Housekeeping Commands  
156 Access Commands  
158 Sequential Text File Commands  
161 Random-Access Text File Commands  
164 Machine Language File Commands
RADIO AND TELEVISION INTERFERENCE

The equipment described in this manual generates and uses radio frequency energy. If it is not installed and used properly, that is in strict accordance with our instructions, it may cause interference to radio and television reception.

This equipment has been tested and complies with the limits for a Class A computing device in accordance with the specifications in Subpart J of Part 15 of FCC rules. These rules are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that the interference will not occur in a particular installation.

You can determine whether your computer is causing interference by turning it off. If the interference stops, it was probably caused by the computer. If your computer does cause interference to radio or television reception, you can try to correct the interference by using one or more of the following measures:

- Turn the TV or radio antenna until the interference stops.
- Move the computer to one side or the other of the TV or radio.
- Move the computer farther away from the TV or radio.
- Plug the computer into an outlet that is on a different circuit from the TV or radio. (That is, make certain the computer and the TV or radio are on circuits controlled by different circuit breakers or fuses.)

If necessary, you should consult your dealer or an experienced radio/television technician for additional suggestions. You may find the following booklet prepared by the Federal Communications Commission helpful:

"How to Identify and Resolve Radio-TV Interference Problems"

This booklet is available from the U.S. Government Printing Office, Washington, DC 20402, Stock number 004-000-00345-4.

PREFACE

This manual has two primary functions. The first is to teach you how to use the DOS (Disk Operating System); the other chapters of the manual are examples to accompany explanations of how the various DOS commands work. The second function of the manual is to serve as a reference guide to DOS. The Appendices, Quick Reference Card, and the Indexes were compiled with this function in mind.

To use an Apple Disk II, you need an Apple II computer with at least 16K of memory -- but 32K is recommended, since the 16K system allows little memory space to store programs. In fact, the MUFFIN program and the FDD program on your System Master Diskette need 32K of memory to run. For using Apple Disk II with AppleSoft on cassette tape or on diskette, your computer must have at least 32K of memory.

The Apple Disk II is a "floppy" disk unit which allows you to store and retrieve information much more quickly and conveniently than you can with tape. The information is stored and retrieved from a "diskette", a small (about 3-inch diameter) specially coated plastic disk which is permanently sealed in a square plastic case.

One of the most important advantages to using Disk II is that information is stored and retrieved by a name under which it is filed. A program that catalogs phone numbers might be named with an instruction such as

SAVE PHONE NUMBERS

and retrieved with an equally simple command. The name PHONE NUMBERS under which the program is stored is a file name.

The programs that automatically keep track of files, save and retrieve information, and do a multitude of other housekeeping tasks are called the Disk Operating System, usually shortened to "DOS". Some people say "dose" and others say "do es oh ess". Learning to use DOS and the disk consists of learning a few special DOS commands described in this manual. These commands can be used as extensions to either AppleSoft or Integer BASIC or machine language programs.

At some places you'll see the symbol

preceeding a paragraph. This symbol indicates an unusual feature to which you should be alert.

The symbol

precedes paragraphs describing situations from which BASIC may be unable to recover. You will lose your program, probably have to re-start DOS, and may have to re-start BASIC.
*** NOTE ***

This manual applies to versions of DOS that operate in 16 sectors. That is, DOS version 3.3 and subsequent. If you have an earlier version of DOS, it will operate in 13 sectors. To update your 13 sector DOS to 16 sectors, change the PROMs on your disk controller card as instructed in Appendix H, and use the MUFFIN program to move your data to 16 sector diskettes. To boot a 13 sector diskette on your 16 sector system, use the BASICS diskette that comes with your Disk II or your Disk II update kit.

Under some circumstances, a diskette converted by MUFFIN may turn out to be un-bootable. This happens when the program on the diskette expects to find instructions at specific locations on the diskette itself. Most programs don’t fall that category, but a few do. Remember, MUFFINing a diskette won’t harm your 13 sector diskette, and you can always boot 13 sector diskettes using BASICS.
UNPACKING

Your Disk II system consists of several items. Among these you will find:

1) The disk drive (the main box).
2) A printed-circuit card (the controller card) that plugs into the Apple II.
3) A flat ribbon cable, already fastened to the disk drive, for connecting the disk drive to the controller card.
4) A "SYSTEM MASTER" diskette.
5) A "BASICS" diskette.
6) This manual.

If you have purchased a drive only (for example, as a second drive for your controller card) your system will not include all of the above items.

Save the packing material in case you wish to transport your disk — or in the unlikely event you must return it to your dealer or to the factory for service.

*** Special Note ***

Before connecting or disconnecting ANYTHING on the Disk II or Apple II TURN OFF THE POWER.
This is a must.

CONNECTING THE CABLE

In use, the disk drive will be connected to the controller card by the flat, ribbon-like cable. One end of this ribbon cable is already fastened to the disk drive. If this is your first disk drive, the connector at the end of the ribbon cable from this drive should be attached to the upper set of pins on the controller card. This set of controller card pins is labelled "DRIVE 1".

*** Caution ***

If the cable from the disk drive to the controller card is not plugged into the controller card correctly, considerable physical damage can be done to the disk drive unit and its electronics. To assure correct assembly, be sure to plug the ribbon cable into the controller card before installing the controller card into the computer. Two installation tips follow. First, don't jam the cable between the connector and the controller card. When the cable is plugged into the controller card correctly, the cable should exit from its connector on the side of the controller card, as shown in the photograph. Second, make sure that all the pins of the controller card's connector go into the matching holes in the

ribbon cable's connector. By making the connection before installing the card, you can actually see that all the pins are going into the holes correctly.

"Connecting the Cable to the Controller"

If you are installing a second disk drive, you should connect the ribbon cable from the second drive to the lower set of pins on your controller. This set of pins is labeled "DRIVE 2". Take the same care attaching this connector as you did with the first.

If you are updating an earlier version of DOS to sixteen sectors, you will need to change two prongs on your existing controller card. Read Appendix H to find out how to do this.

INSTALLING THE CONTROLLER

To install the Disk II controller card, which you have already connected to the disk drive via the ribbon cable, you will simply plug the controller card into a socket inside the Apple II, as follows:

1. Turn off the power switch at the back of the Apple II. This is important to prevent damage to the computer. If the power is on, removal or insertion of any card could cause permanent damage to both the card and the Apple II.

2. Remove the cover from the Apple II. This is done by pulling up on the cover at the rear edge (the edge farthest from the keyboard) until the two corner fasteners pop apart. Do not continue to lift the rear edge, but slide the cover backward until it comes free.

3. Inside the Apple II, across the rear of the circuit board, there is a row of eight long, narrow sockets called "slots". The leftmost one (looking at the computer from the keyboard end) is slot #8, and the rightmost one is slot #7. Locate slot #6, one socket to the left of the rightmost socket. The controller card may be placed in any slot except slot #4, the leftmost. However, Apple's standard location for the disk controller card in slot #6, and most Apple software (and this manual) is written with that location in mind.
4. **BE SURE THE POWER IS OFF BEFORE YOU INSERT OR REMOVE ANY CARD FROM THE COMPUTER.** Insert the "fingers" portion of the controller into slot #6. The "fingers" portion will enter the socket with some friction and will then seat firmly. Since the fingers make electrical contact, it is a good idea to keep your fingers from touching them. Before installation, you may wish to use rubbing alcohol to clean the fingers on the board (and, optionally, your own fingers if you're so inclined).

![Inserting the Controller Card](image)

5. Adjust the ribbon cable so it lays flat and passes over one of the areas between the vertical openings in the back of the Apple II case, as shown in the drawing. When the lid is installed it will clamp down the cable and act as a strain relief.

![Cable Placement](image)

6. Replace the cover of the Apple II; remember to start by sliding the front edge of the cover into place. Press down on the two rear corners until they pop into place.

7. The Disk II controller is installed, and the Apple II may now be turned on. Place the disk drive in a convenient location, usually alongside of or on top of your Apple II.

---

**INSTALLING MULTIPLE DISK DRIVES**

Each controller card can be used with two disk drives, one attached to the upper set of pins, labeled "DRIVE 1", and the second attached to the lower set of pins, labeled "DRIVE 2". Your first disk drive should be attached to the DRIVE 1 pins and the second to DRIVE 2 pins on the card in slot #6. The third and fourth drives should be attached to the DRIVE 1 and DRIVE 2 pins, respectively, on a card in slot #5, the fifth and sixth drives attach to the DRIVE 1 and DRIVE 2 pins on a card in slot #4, and so on.

If you have multiple drives, it is a good idea to label the front of each drive with its slot and drive number since your programs will refer to the disks by those numbers.

---

**CARE OF THE DISK II AND DISKETTES**

The Disk II drive, unlike the Apple II, is a mechanical device, with motors and moving parts. Therefore it is somewhat more delicate than the computer. Rough handling, such as dropping the drive, or having things drop on it, can cause it to malfunction. The drive should not be placed beside or on a TV set, since the strong magnetic fields put out by TVs may cause damage to the magnetic properties of the drive. If in doubt, locate disk drives at least 2 feet from any TV set.

Each diskette is a small (about 5-inch diameter) plastic disk coated so that information may be stored on and erased from its surface. The coating is similar to the magnetic coating on recording tape. The diskette is permanently sealed in a square black plastic cover which protects it, helps keep it clean and allows it to spin freely. This package is never opened.

The term "floppy" comes from the fact that the diskette is flexible. Older computer information storage devices that worked on similar principles used rigid disks. While the diskette (and its plastic cover) are somewhat flexible, actually bending the diskette can damage it. The diskette cover contains both lubricants and cleaning agents to extend trouble-free operation - treat covers with respect.

Never let anything touch the brown or gray surface of the diskette itself. Handle the diskette by the black plastic cover only. When a diskette is not in use, keep it in the paper jacket that it came in. These pockets are treated to minimize static build-up which attracts dust. It is best to store diskettes vertically when they are not in use. Vinyl notebooks especially made for this purpose are convenient.

Diskettes hold a tremendous amount of information; a single diskette can hold over 1,146,000 bits of information. An individual bit of information, therefore, occupies a very small portion of the diskette. An invisible scratch on the surface of the diskette, or even a fingerprint, can cause errors. Do not place diskettes on dirty or greasy surfaces; do not let them collect dust.
To write on a diskette label, use a FELT TIP pen. Do not press hard. It is best to write on a label attached to a diskette, but to write on the separate label, then attach it to the diskette.

Keep diskettes away from magnetic fields. This means to keep them away from electric motors and magnets; they should not be placed on top of electronic devices such as television sets. They may be temporarily laid on the Apple II or the Disk II.

Diskettes are sensitive to extremes of temperature. Keep diskettes out of the sun, and away from other sources of heat that can cause them to warp and/or lose data. On hot days, car trunks (or dashboards) can be diskette killers. Diskettes operate satisfactorily up to 125 degrees Fahrenheit (51.7 Celsius), which is not very hot. The first evidence of heat damage is a warped or bent black plastic cover.

With reasonable care a diskette will give you an average life of 48 hours -- which is a lot, when you consider the few seconds it takes to LOAD most programs. With just a little bit of carelessness, a diskette may give you no service at all.

**INSERTING AND REMOVING DISKETTES**

Using a disk drive is far quicker and easier than using a cassette recorder, however some care is necessary to protect the diskette. The drive itself must also be handled with some care. The drive door is opened by pulling outward on its bottom edge. The diskette is then slipped into the slot with the label upwards, as shown in the photograph. The edge of the diskette with the oval cutout in the diskette's square plastic cover should enter the drive first. The edge of the diskette with the label should enter the drive last.

**A Good RULE OF THUMB**

Hold a diskette with your right thumb over the label; that pretty much insures the correct orientation when you put the diskette in the drive.

---

*Inserting a Diskette*

Push the diskette gently until the diskette is entirely into the drive. Do not bend the diskette: If it is pushed in too hard, the diskette can be permanently damaged. Close the drive door by pushing it down again.

The two metal fingers (which can be seen inside the slot when the drive door is closed) should just clear the diskette as the door closes.

A diskette is removed by opening the drive door and pulling the diskette carefully out of the drive. The act of opening the disk drive door lifts the "head" off the disk. If you plan to leave an unused diskette in a drive for several hours, it's a good idea to open the door so the head won't rest on the diskette.

**A Good RULE OF THUMB**

Hold a diskette with your right thumb over the label; that pretty much insures the correct orientation when you put the diskette in the drive.
BACKGROUND

Learning to use the disk and its operating system consists of learning a few special instructions, several of which are straightforward extensions of familiar BASIC instructions. This manual assumes that you're familiar with the Apple I and feel comfortable writing simple BASIC programs.

To learn how to use the Apple II and Integer BASIC, consult the Apple II BASIC Programming Manual (Apple Product PAIL0006X). To learn how to use Aplosoft BASIC, consult the Aplosoft II BASIC Programming Reference Manual (Apple Product PAIL006). The Appleloft manual assumes you're already familiar with the Apple II and simple BASIC programming. If you're not familiar with either manual, we will wait here while you learn about the Apple II, before going on to learn about DOS.

Throughout the manual are listings of programs that illustrate how to use DOS. Most of these programs are in Aplosoft; a few are in Integer BASIC. Sometimes the changes needed to convert an Aplosoft program to Integer BASIC are mentioned; other times, they are not. Consult Appendix M in the Aplosoft manual for details on the differences between programs written in Integer BASIC and Aplosoft BASIC.

A little bit of hands-on experience is worth a lot of reading. Once your disk drive is hooked up and the computer is turned on, follow each of these descriptions by actually trying out the procedures on your Apple II.

Put the Apple II into BASIC — either Integer BASIC or Aplosoft. Place the System Master diskette into the drive. The diskette should be labelled DOS—PPZ—X3. The last two digits are indicated by X's, since it doesn't matter what they are. If you have more than one drive, use Drive 1. This section of the manual only deals with one drive and assumes that you've followed the standard conventions, putting the controller into slot #6.

With the disk drive attached, and the diskette in the drive, and the disk drive door closed, you will find that the Apple II operates just as it did without the disk. Nothing is changed. It is as if the disk drive were not there. And, as far as the Apple II is concerned, the disk drive is not connected yet. A special command must be given to inform the computer that the disk drive and the new DOS instructions are available.

Even though DOS commands look like BASIC commands, they do not always follow the same rules. For example, multiple DOS commands cannot be put on one line, separated by commas. The SYNTAX ERROR message results.

SPECIAL KEYS

Sometimes this manual uses the early brackets ( and ) to enclose the names of special keys which you are supposed to press on the Apple II keyboard.
In the rest of this manual, when you are to re-start the DOS in this manner we will simply say: "boot the DOS" or "boot the disk!" Both expressions (very popular among computer users) mean the same thing. "Boot" is short for the word "bootstrapping" and the term is from the expression "to pull oneself up by one's bootstraps". In any case, it does not mean to kick the disk, even if you do feel in such a mood from time to time.

Now try booting DOS from your System Master diskette. Start by putting your Apple II in BASIC -- either Integer BASIC or AppleSoft will do. Be sure the diskette is properly inserted. Next type PR#6 and press the RETURN key as usual. From now on, it will be assumed that you will press the RETURN key after each instruction.

Once you press the RETURN key, the red "IN USE" light will come on, the disk will make whirring and clacking noises (don't be alarmed -- it's not getting ready to fly away) and in less than 10 seconds, a message will appear. The message should be similar to the following:

If you now try to use BASIC, you will find that most commands still operate normally and, aside from the message suddenly appearing, the Apple II seems unchanged. What has happened in this: a few new commands have been introduced, and a few old ones have new capabilities. Two changes have been made that are not obvious, however:

1) The HIMEM pointer to the highest memory location you may use has been reset to accomodate the DOS program.
2) Your Apple II may have lost some of its high-resolution graphics capabilities, depending on the amount of memory in your computer.

** Versions of DOS that use 13 sectors can't be booted when the system expects 16 sector diskettes. (The diskette spins and hiccupps, but nothing comes out.) To run a 13 sector diskette, update it to 16 sectors with the MUFFIN program. You can also use the BASICS diskette to boot 13 sector diskettes on your sixteen sector system. See Appendix I to learn how.

IF BOOTING DOESN'T WORK

If you can't successfully boot your System Master diskette, re-read the manual carefully -- that cures 90% of all problems.

This isn't likely, but if your unit was shipped in a Sherman tank or some such, the connectors inside the disk drive may have worked a bit loose. If you are at all squeamish about handling the insides of your drive, your dealer will be glad to check it out.

If you enjoy getting your fingers into the works, you can turn the computer off, and disconnect the drive from the controller. Loosening the four screws on the bottom of the drive allow the mechanism to slip forward out of the case. Tighten the connectors by pushing them gently onto the circuit boards.

Re-assemble the unit and it will probably now work. If this first aid doesn't work, see your dealer. Don't make any adjustments.

INITIALIZING NEW DISKETTES

The System Master that comes with this manual is a very special diskette: it contains programs that allow you to copy an entire diskette, update any diskette that has an earlier version of DOS, and more. Programs that demonstrate various capabilities of DOS are also included on the diskette and discussed in the manual.

Take the System Master diskette from the drive, and replace it with the other blank diskette supplied with your drive. Now try an experiment. Get BASIC going, then type PR#6 and watch what happens. The red "IN USE" light comes on, and the disk drive makes a few clackety noises, then it just keeps whirring softly and quietly and it doesn't stop. You'll have to press the RESET key to stop it (normally, this is a BAD idea, but these circumstances aren't normal). It's a good idea to open the disk drive door before pressing the REST key, since that lifts the head of the disk drive off the surface of the diskette.

What happened was that your Apple II went on a fruitless unending search for information on a blank diskette (on a clear disk you can search forever...). When a new diskette is manufactured, it contains no information at all, like a blank tape purchased for a tape recorder. To operate in the computer, there must be special information placed on the diskette: the diskette must be initialized.

If you've been keeping up with the hands-on part of the example, your blank diskette is in the drive and you just pressed the REST key. Now take out the blank diskette, replace it by the System Master diskette, and close the door of the drive. Get the computer into BASIC and type PR#6 again. You should again get the message you got before you booted. Once move the DOS commands have been added to BASIC.

The INIT command can be used to initialize a "slave" diskette. Slave diskettes are memory-size dependent: the size of the system which initializes the diskette determines the size of the system which can use the diskette. If a slave diskette is created on a 32K system, then it can only be used in a system with 32K or more memory. On larger systems, only 32K of memory will be used. After initializing a slave diskette, you can use the MASTER CREATE program (see Chapter 3) to transform your slave diskette into a "master" diskette whose DOS is self-relocating so that memory is used efficiently. The MUFFIN program will allow you to transfer the contents of your 13 sector diskettes to 16 sector diskettes.

The INIT command requires the use of a BASIC program called the "greeting program since it greets your each time you boot the diskette the program will be run automatically. The greeting program is commonly named "HELLO" but you could call it "BONJOUR" or "BUENOS DIAS" or whatever you like. It helps keep life simple to use a standard name for greeting programs as you INITIALIZE diskettes.
Here's a step-by-step guide to INITializing a slave diskette. We assume DOS is already booted as described above.

1) Remove the System Master from your disk drive and replace it with a blank diskette.

2) Type NMI, then type a greeting program. Here is a simple sample of a greeting program:

```
REM GREETING PROGRAM
10 PRINT "SLAVE DISKETTE CREATED ON THIS SYSTEM"
20 PRINT "YOU HAVE 10 SECONDS TO DO THIS"
30 GOTO 20
```

You should supply your own name, system size, the current date and other information to help you quickly and easily determine the diskette's history and slave/master status. You may RUN the program to see if it does what you expect.

3) Once the program is satisfactory, type this instruction:

```
INIT HELLO
```

When you press the RETURN key, the diskette will spin for nearly a minute, making clacks and little whispery noises every now and then. The appropriate prompt character (e.g., I for Applesoft) will be displayed when initialization is complete.

4) When the disk quietens down and the IN USE light goes off, remove the diskette and label it. The label should say something like

```
32K SLAVE DISKETTE CREATED 8 AUGUST 1980
```

Put aside the system master diskette supplied by Apple Computer. Put it where it won't be damaged by heat, physical stress (kids, dogs?) or magnetic objects. And where it won't get lost. It should be treated especially carefully, since it contains many useful programs.

Once a diskette has been INITialized, it will be referred to as a slave diskette. To label your slave diskette, you had to take it out of the drive. Put it back in and try booting it: the message in your PRINT statements should appear. If you followed the model given above, the screen should say:

```
SLAVE DISKETTE CREATED ON THIS SYSTEM
YOU HAVE 10 SECONDS TO DO THIS
```

Since the once-blank diskette now can boot, you know that it has been INITialized correctly. From this point on you will use the newly INITIALIZED slave diskette for experimentation. You cannot do some of the procedures to be demonstrated below on the System Master, because the diskette is "write protected", as discussed in Chapter 4.

If you have purchased additional blank diskettes, it would be a good idea to INITialize a few of them now.

LOAD-ING AND SAVE-ING WITH DOS

Boot the system with your initialized diskette. Type NMI to make sure no programs are in memory. This will erase your greeting program (which is LOADed and RUN when you boot DOS) from memory (but not from the diskette).

Now type this simple program:

```
5 REM COUNT PROGRAM
10 FOR I = 1 TO 10
20 PRINT I,
30 NEXT I
40 END
```

RUN it once or twice to make sure that it works as you expect. In Applesoft, when the program is RUN you'll see this:

```
1 2 3
4 5 6
7 8 9
10
```

For reference purposes, call this program ONE TO TEN, since it counts from one to ten. To store this program on the diskette, type the instruction

```
SAVE ONE TO TEN
```

When you complete the command by pressing the RETURN key, the disk will whirr for a few seconds, and the program will be saved.

If you had typed SAVE without any name, the program would have been saved on cassette tape, as usual (assuming you had operated the tape recorder as described in the BASIC Programming Manual).

To prove that the program has been SAVED on diskette, do the following. First, type LIST

then RUN

to see the program is still in memory and still operates properly. This demonstrates that using DOS to SAVE a program on a diskette doesn't affect the program in any way.

Now type NMI

then LIST

There will be no program left at all -- it disappeared when you typed NMI.

To really make sure the program is dead, turn off the computer. You can even take the diskette out and put it (gently) back in again. Turn the computer back on again, get into BASIC and boot the DOS. Type NMI (which erases the HELLO program), and then LIST. Nothing there? Right.
Now type
LOAD ONE TO TEN
and the disk will whirr for about two seconds. LIST the program: it is
revived, RUN it, and you will find it in perfect health. That is all there is
to SAVING and LOADING programs from disk: it’s just like using the
cassette tape except that a file name is used, and it’s faster.

CATALOG

You stored the program ONE TO TEN on your diskette. Actually, you had
already stored another program. To see what programs are stored on a
given disk, type the command
CATALOG
and a list of all the programs on the diskette will appear. Right now your
diskette’s catalog should look like this, if your programs were written in
Integer BASIC:

I 001 HELLO
I 002 ONE TO TEN

The letter "I" in the left column means that the programs are in Integer
BASIC; before names of Apple-soft programs you’ll see an "M". Besides BASIC
program files, there are also other kinds of files that can be stored, and
they will be explained in Chapters 6 through 9. The numbers after the
file-type letter represent the length of the stored program. In this case,
001 diskette "sectors" were required to store the program. Each
diskette sector can store up to 255 bytes of information. The shortest
possible file, an empty text file (see Chapter 6), requires 001 sector to
record certain "housekeeping" information. In all, a diskette can store
496 sectors of programs and other files. Lastly, each entry in the
catalog contains the name of the program. See Appendix C for details on
how information is stored on the diskettes.

When a file exceeds 255 sectors, the length reported for that file by
CATALOG starts over again at 000.

There is no way to tell from looking at the CATALOG which program is the
greeting program. So it helps if you always give the same name to your
greeting program.

Sometimes you’ll have more programs on a diskette than will fit on the TV
screen at one time. CATALOG will cause the first 18 programs to be
listed. When you’re ready to see the other programs on the diskette,
press any key except the RESET key, CTRL key or the SHIFT keys.

WHAT’S IN A NAME?

File names must be from 1 to 30 characters in length; DOS will truncate
longer file names to 30 characters. A file name must begin with a letter.
Any typable character except the comma (,) may appear in the name.

Here are some legal file names:
SOMNAMBULISTICS
ONE TO TEN
HITRES 24
THE QUALITY OF MERCY: UNSTRAIGHTED

Here are a few names that will not work (and reasons why):
1 TO 10
BEGIN, ABE
INEXP-EXCESS VERBIAGE DISQUALITIES NAMES
(will be cut to 30 characters)

Although the name of the last file will be cut to 30 characters when
displayed by CATALOG, you can, if your fingers can take it, type the
entire name when LOADING or RUNNING, and all will work correctly.

Every line in the catalog represents a "file". The BASIC program you
stored is an example of a file. The rules given here for file names also
apply to the names of programs.

If a control character is accidentally (or even purposefully) typed into a
name, that character will not appear on the screen when you get a catalog.
For example, if you type (CTRL)E instead of plain "E" in the name
"ACATHA", the catalog listing would appear to be
ACATHA

However, if you tried to LOAD that file by typing
LOAD ACATHA
the computer would reply
FILE NOT FOUND
even though the name you typed seemed to be identical to the name in the
catalog. So be careful: don’t inadvertently put control characters in
file names. (Although, heh heh, it’s a clever way to keep you out of my
bank records if all files have secret control characters embedded in them...)
The File Names section of Appendix F contains tips on how to find out
what control characters are imbedded in file names.

RENAME-ING FILES

For one reason or another, you’ll occasionally want to change the name of
a file. Suppose you get tired of typing the file name ONE TO TEN and
decide to call it the file COURT. Just type
RENAME ONE TO TEN, COURT
and after a moment of whirring you’ll again see the BASIC prompt
character. Now type
CATALOG
to verify that all went as planned.

The RENAME command does not check to see whether the new name you’re
using already exists or not, so it’s entirely possible that you can RENAME
until all files on a diskette have the same name...a most undesirable and
confusing situation that is best avoided.
DELETING FILES

It is easy to remove files from the diskette. Type

```
CATALOG
```

to see the two files that are on your diskette. You type

```
LOAD COUNT
```

(assuming you changed the file name as shown above) to get that program

into memory. Delete this program from the diskette by the instruction

```
DELETE COUNT
```

and test that your deletion has worked by typing

```
CATALOG
```

Only the greeting program -- probably called HELLO -- is left. Since the
program COUNT is in memory (that's why you LOADED it), you can place it
back onto the diskette with the familiar command

```
SAVE COUNT
```

Take a look at the catalog to see that the program is again on the

diskette.

If you try to DELETE a file that's not on the diskette, you'll receive the

```
FILE NOT FOUND
```

message.

RECOVERING FROM ACCIDENTAL RESETS

Suppose you're without the DOS in either Integer BASIC or Applesoft. (If
Applesoft is in firmware, we assume the switch on the card is set for
Applesoft.) If you accidentally strike the RESET key, you can recover with
your program intact by using CTRL-C. The DOS also has recovery procedures
that will usually preserve your program and data.

If you are working with an old Monitor ROM and have already booted DOS, and
then press RESET, you get the Monitor (A) prompt. To return to DOS and
the BASIC you left, type

```
3085
```

Remember that's a zero, not the letter O, between the letters D and G.

If recovery to DOS does not work, and the program still LISTS, all is not
lost: save the program on tape (you did remember to keep your tape drive
for just such an emergency, didn't you?). Then at your leisure you can
boot DOS, LOAD the program from tape, and SAVE it on a diskette.

If you accidentally (or intentionally) hit the RESET key while the disk's
red "IN USE" light is on, the information on your diskette may be
clobbered. Problems are most likely to occur if this happens when you're
putting information onto the diskette using a SAVE, RS, or WRITE
command. In the event that it is clobbered, you probably won't be able to
recover your programs from the diskette. If nothing else works, you can
re-initialize the diskette and use it again, but INITIALIZING destroys all
the files on the diskette.
CHAPTER 3
EXERCISING OPTIONS

23 Drive, Slot, and Volume Options
24 Syntax
24 INIT
25 LOAD, RUN, and SAVE
26 DELETE
27 A Scenario: boot, CATALOG, SAVE, RUN, and DELETE
28 Moving Between Languages: FP and INT
29 Use of DOS From Within a Program
DRIVE, SLOT AND VOLUME OPTIONS

Most DOS commands allow you to specify a number of options, such as which disk drive you are using, which slot contains the disk controller for that drive, and a "volume number" for the disk.

The disk drive option allows you to operate with more than one drive. Each controller has the ability to control one or two disk drives. Normally, instructions refer to drive 1. This is the default drive selection; if you don't specify a drive, drive 1 will be used. If you wish to specify drive 2, you use the notation D2 or D0, depending on the file name or other disk options by a comma. For example, to initialize a diskette in drive 2, you could use the instruction

```plaintext
INIT HELLO, D2
```

After drive 2 has been specified, all further disk commands refer to drive 2 until drive 1 is again specified. Drive 2 is now the default drive. After the above initialization, the command

```plaintext
CATALOG
```

will list the files stored on the diskette in drive 2. To specify drive 1, you use the notation D1 separated from the file name by a comma. For example,

```plaintext
CATALOG, D1
```

will show you the contents of the diskette in drive 1, and change the default drive number back to 1.

If more than two drives are in use, then additional controllers are required. These are placed in different slots than the first controller (which is customarily in slot number 6). You can specify slot n (where n is a digit from 1 to 7) with the notation Sn separated from the file name or other disk options by a comma. For example, to initialize a diskette in drive 1 attached to a controller in slot 5, you would use the instruction

```plaintext
INIT HELLO, S5, D1
```

The file name must come first, but order of the options is not important.

The default slot number is the one you used when booting the DOS. Once a different slot number has been specified, it becomes the default slot number until it is explicitly changed.

After using a DOS command with a Slot parameter naming a slot that doesn't contain a disk controller, you get an

```plaintext
I/O ERROR
```

message, and all appears to be fine. But DOS now thinks the default slot number is the bad slot number, and that the disk that isn't connected to that slot is still running. Even if the next DOS command specifies the right slot, it waits in limbo forever for the non-existent disk to stop running the last command. If you have no program in memory that you care to save, simply re-boot DOS. To recover with your program intact, do this:

1) Reset the default slot by typing
```
CATALOG, S0
```

where s is the correct slot number,

2) When the system hangs, press the RESET key.

3) Type

```plaintext
DOS
```

and all should be fine again.

DOS must be booted from a diskette in Drive 1, not Drive 2.

The volume number option can be used to protect diskettes from being accidentally written over. For example, suppose you have a diskette-based inventory system, where each month's records is on a different diskette with a unique volume number. Then when you go to enter information for the month of January, you must be sure to specify the correct volume number. Otherwise, the information won't be written to the diskette and you'll get a

```plaintext
VOLUME MISMATCH
```

message.

A "volume number" may be assigned to a diskette when it is INITIALIZED, using the notation Vn separated from the file name or other disk options by a comma. For instance, to initialize a diskette using the name "START UP" for the greeting program (the program that is run each time the diskette is booted), where the diskette is in drive 2 of a controller in slot 5, assigning the diskette a volume number of 128, you would use the command

```plaintext
INIT START UP, D2, S5, V128
```

The volume number of a diskette may not be changed without re-INITIALIZING the diskette.

The drive number, slot number and volume number options may appear in any order. The above command is equivalent to

```plaintext
INIT START UP, V128, D2, S5
```

and to

```plaintext
INIT START UP, S5, V128, D2
```

and so on.

The volume number of a diskette must be an integer from 1 through 254. If no volume number is specified with INIT, a default volume number of 254 is assigned to the diskette.

The command

```plaintext
INIT HELLO, V8
```

does not give any message, but assigns the diskette the default volume number 254.

All DOS commands can specify the volume number, if you wish DOS to check that the volume number on the diskette agrees with the V option. If you do not specify any volume number, or if you specify volume zero, or if you type "V" without a number, DOS will ignore the diskette's volume number. If you accidentally specify an incorrect volume number, the system will reject it with the message

```plaintext
VOLUME MISMATCH
```
Volume mismatch errors cannot occur when you ask to see the CATALOG. In case you wish to know the volume number of a diskette, it is given at the head of the CATALOG listing.

Additional discussion of options is found where each command is introduced. Also, the information is concisely summarized for each command in the Command Summary Appendix and on the Quick Reference Card accompanying this manual. The following section explains how to interpret these concise summaries.

SYNTAX

Syntax refers to the structure of a computer command, the order and correct form of the command's various parts. A simple notation is used to describe the syntax of each DOS command. Items in brackets ([ ] and { }) are optional; optional parts of a DOS command may be specified in any order. CAPITAL letters and commas must be typed as shown; lower case letters stand for items you must supply. In specifying the syntax for DOS commands,

- f stands for a file name
- d stands for drive number -- either 1 or 2.
- s stands for slot number -- 1 through 7.
- v stands for volume number -- 1 through 254, usually.

A diskette's volume number may not be 0. Specifying a volume number of 0 in a disk command is a "wild card" that tells DOS to ignore the volume number on the diskette.

Additional abbreviations used in this manual are summarized at the start of the Command Summary Appendix.

Any numerical constant (the drive number, volume number, etc.) in a DOS command can be expressed in hexadecimal notation by preceding the hex digits with a dollar sign. If you don't know what hexadecimal notation is, ignore the preceding statement -- you needn't understand hex notation to understand this manual.

INIT

The syntax for the INIT command is

INIT f [{,v} {,s} {,d}]

where the brackets indicate options which may or may not be included. The example

INIT HELLO, V17, D2

can be interpreted as follows.

The command name "INIT" is in upper case, and must be typed exactly as shown. The lower case "f", for file name is replaced by the legitimate file name "HELLO". Next the optional volume number is indicated: first comes a comma, then the upper case "v". The "v" for volume number was arbitrarily replaced by 17 for this example. The brackets around ",s" indicate that specifying the slot number is optional for the INIT command.

In this example it's omitted so DOS will use the default slot number. The drive option is included; the comma and upper case "f" must be as shown; the lower case "d" is replaced by 2 in this example.

LOAD, RUN AND SAVE

LOADing, RUNNING and SAVEing programs on the disk is similar to the corresponding operations using the cassette (except that programs are referenced by file name). Everything goes at least ten times faster, and you never need to press buttons to play, record or rewind. It is all automatic. There are many additional abilities that the disk brings as well, such as the catalog of programs and the automatic running of programs without user intervention. Saving data (on text files -- see Chapter 6) is also very easy. The FIB program described in Appendix J offers you some additional ways to manipulate files.

It's a good idea to hang on to your cassette tape system for trading programs and as back-up storage for vital programs and data (although experience shows that disk storage is even more reliable than cassette storage of programs and data).

If you have a program in BASIC, and you wish to call it HENRY, then the command

SAVE HENRY

will save it on the diskette. If you have more than one drive, HENRY normally would be saved on the drive from which you booted DOS (the default drive, unless you specified a different drive after booting). You can specify drive number, volume number and slot number as with the INIT command. For example, to SAVE a file called AGATHA on drive 1 of the controller in slot 2, where the volume number of the diskette is 214, you could use the command

SAVE AGATHA, D1, S2, V214

As before, the three options can be put in any order. If you had omitted the volume number option, AGATHA would have been saved just the same, bless her, but DOS would not have checked that the diskette was volume 214.

Program names are file names, and must follow the file name rules: they may be up to 38 characters long, and must start with a letter. They may include any characters you can type except commas or control characters. Here are some valid names for files:

CHECKBOOK
THE QUALITY OF MERCY
RIRES36
NOW! HEAR THIS!

To LOAD a program named AGATHA, use the command

LOAD AGATHA

and the program of that name, if there is one in the catalog, will be loaded. To test if AGATHA is loaded, see if she can walk a straight line.

If you want AGATHA to RUN after she's LOADED (poor thing) you can, of course, use the commands

LOAD AGATHA
RUN

But there's a way to do it in just one step:

RUN AGATHA

is a DOS command that first LOADs the specified file, then RUNs it.

Here's the syntax for the SAVE and LOAD and RUN commands:

SAVE f [ ,Sa ] [ ,Dd ] [ ,Vv ]
LOAD f [ ,Sa ] [ ,Dd ] [ ,Vv ]
RUN f [ ,Sa ] [ ,Dd ] [ ,Vv ]

Examples follow:
SAVE OUR HAPPY HOME, D1, S7
LOAD UP
RUN AMOK, S7

If, when you try to SAVE a program, you get a SYNTAX ERROR message, either you have made a typing error, or DOS isn't booted. First, try re-typing the command. If DOS was originally booted, use 3000 to try to recover. If DOS isn't booted -- DON'T BOOT IT. Booting DOS will erase any program in memory. First, save the program on tape, using the usual cassette SAVE command. Now boot DOS. Next, use the usual cassette LOAD command, to bring the program back into your APPLE II's memory from the tape. Now you will be able to SAVE it on disk.

If a diskette is bad (perhaps someone tried to staple it into a notebook), or if the diskette is not initialized, or if there is no diskette in the drive, or if the door is open, the message I/O ERROR (I/O stands for Input or Output) will appear when you try to SAVE or LOAD using DOS. Check all the items listed, and correct the problem. You don't need to re-boot DOS. Try again.

If you use the command LOAD HENRY and HENRY is not the name of a program on the diskette in the drive, then you get this message:
FILE NOT FOUND

Look at the diskette's catalog to find the program's exact file name. All characters and spaces must be typed exactly as they appear in the file name shown in the catalog.

DELETE

To eliminate any file that you would rather not have on your diskette, the command DELETE can be used. The syntax is:

DELETE f [ ,Sa ] [ ,Dd ] [ ,Vv ]
MOVING BETWEEN LANGUAGES: FP AND INT

Suppose you’ve been using Integer BASIC, and you decide to write a program in APPLESOFT, or to use the computer as a calculator with floating point numbers (numbers with decimal points). To invoke APPLESOFT without erroring DOS, type FP (that’s all there is to it) and in a few seconds APPLESOFT will be up and running. The FP stands for “Floating Point”, of course. (If for some reason Applesoft isn’t available – it’s not in firmware or on the diskette that’s in use — then the message LANGUAGE NOT AVAILABLE will be displayed.) The syntax for the command is FP [/S] [/D] where the optional Slot and Drive parameters allow to to specify the drive containing Applesoft on a diskette.

If you’ve been using APPLESOFT and DOS, you can type INT (for “Integer BASIC”) to return to Integer BASIC with DOS intact. The syntax for this command is simply INT without any parameters. You’ll generate a SYNTAX ERROR message if you try to use the D or S parameters with INT.

If you type INT while in Integer BASIC, you will lose any program in memory. Similarly, if you type FP while in Applesoft, you’ll lose any program in memory.

When you switch from Integer BASIC to Applesoft or vice versa, you’ll lose any program you happen to have in memory.

In addition to moving back and forth between the Apple’s BASICS, you may wish to enter the Monitor and be able to use DOS commands. To do so from either Applesoft or Integer BASIC, type CALL -151 and you should get the Monitor prompt character. * . To return to whatever BASIC you started from with your program and DOS intact, type TRG.

From the Monitor, you may also type INT to return to Integer BASIC, or FP to return to Applesoft; in either case, DOS will still work but any program in memory will have disappeared.

If you get a PROGRAM TOO LARGE message when trying to execute an FF command, type INT first, to reset the system. Then type FF.

Even though your diskette contains the integer BASIC program named APPLESOFT, do not type RUN APPLESOFT. If you do, Applesoft will seem to be running fine until you press RESET, say, and try to re-enter Applesoft. Then, since the DOS thinks you are in Integer BASIC (because APPLESOFT was an integer BASIC program), you will be in trouble.

To move the APPLESOFT program from one diskette to another, simply LOAD APPLESOFT from whatever diskette it’s on, then place the diskette you wish to contain Applesoft in the drive and type SAVE APPLESOFT.

USE OF DOS FROM WITHIN A PROGRAM

Very often it’s useful to be able to execute a DOS command from within a BASIC program. For example, you may wish your greeting program on a disk to print out the contents of the disk by doing a CATALOG command. Many DOS commands can be executed from inside a BASIC program. This is done by PRINTing a string that consists of a CTRL-D followed by the command.

Here is an Applesoft program that, if used as a greeting program, will cause the information in the PRINT statements in lines 20 and 30 to appear on the screen, followed by a list of files in the CATALOG.
In DOS commands executed by a program, the D$ must be preceded by a RETURN or it will be ignored. Running this program

```
5 REM TESTCATALOG PROGRAM
10 D$ = "": REM THERE IS AN INVISIBLE CTRL-D BETWEEN THE QUOTES
20 PRINT "TEST";
30 PRINT D$: "CATALOG"
40 END
```

will cause TESTCATALOG to be displayed, since the semi-colon suppresses the RETURN at the end of the PRINT command in line 20. To correct this, and cause the DOS command CATALOG to be executed when the program is RUN, just delete the semi-colon (;) from the end of line 20.

These DOS commands should only be used within programs in a PRINT statement beginning with a CTRL-D:

```
OPEN APPEND
READ WRITE POSITION
```

These DOS commands may be used in immediate-execution mode, and also from within a program using a PRINT command with CTRL-D:

```
CATALOG SAVE BSAYE
SAVE BLOG
LOAD BBUN
RUN EEXEC
DELETE CLOSE
RENAME CHAIN
LOCK and UNLOCK PRF
NON and NOLON INF
```

The DOS command MAXFILES may be used as described above in an Applesoft program, but it must be used in a special way from an Integer BASIC program, as discussed in the section about the EXEC command in Chapter 7.

The DOS command INIT should be used only in immediate-execution mode (dire consequences may result if you ignore this admonition).

Using the right-arrow to copy a BASIC statement containing an invisible control character will erase the control character.
CHAPTER 4
PLAYING SAFE

34 Creating a Turnkey System
35 LOCK and UNLOCK
37 VERIFY
36 Write-Protecting a Disk
37 Protecting Yourself against Disaster
38 Using the CEFF Program
Two ways of protecting you and/or your diskettes against disaster have already been mentioned. Chapter 3 mentions using the Volume option to ensure that you place information on the desired diskette. The use of control characters in file names can also be used as a way of protecting yourself (see Chapter 2, "What’s in a Name?" and also Appendix F, "File Names"). If what appears in the CATALOG as MY BANK ACCOUNT in fact has your initials placed as control characters at some point(s) in the name, then it’s unlikely that anyone else can access the file.

This chapter mentions a variety of ways of protecting you and your diskettes against various undesirable events. You’ll probably find one or more of the techniques useful at one time or another. First, consider making a special purpose "turnkey" system.

CREATING A TURNKEY SYSTEM

Suppose a doctor wants to do the office accounting on an APPLE II. Ideally, the office staff should be able to simply turn on the APPLE II, type

(RESET) 6 (CTRL)P (RETURN)

and immediately be in the midst of the doctor’s accounting program. Since the accounting program would (hopefully) communicate with the user in ordinary English, the staff wouldn’t need to know BASIC or anything else about the APPLE II. The computer would become an accounting system, its internal characteristics unimportant since all the staff needs to know is how to use the accounting program.

This is the essence of a "turnkey" system: from the user’s point of view the computer is a device that does only a particular task, and getting the system started is as simple as turning a key in a lock. In this case, the "key" is simply turning on the Apple’s power switch and pressing five keys on the keyboard. It does not require computer expertise to be able to do that.

You can use the diskette’s "greeting program," named when you INITIALIZED the diskette, to turn your APPLE II into a turnkey system. Let’s say that you wanted the computer to run the COLOR DEMO program (provided on the System Master diskette) every time you booted Disk II. Here’s how:

1) INITIALIZED a blank diskette, as described in Chapter 2.

2) place the System Master diskette in your drive and type RUN COLOR DEMO Once you’re satisfied that the program runs correctly, type (CTRL)C to stop the program and return to BASIC.

3) Put your newly INITIALIZED diskette into your drive. We’ll assume that you called your "greeting" program HELLO when you INITIALIZED the diskette.

4) The program COLOR DEMO is now in memory. When you type SAVE HELLO DOS will erase your original greeting program named

HELLO and save the COLOR DEMO program under the HELLO file name. The COLOR DEMO program is now the greeting program on your diskette.

To check that all works as expected, boot the disk. You should get the same program that you used in step 2.

You’ve just created a turnkey system; whenever that diskette is booted, it will automatically LOAD the COLOR DEMO program and RUN it.

LOCK AND UNLOCK

Sometimes you’ll want to prevent a particular program from accidentally being erased from a diskette: the LOCK command will do this for you.

Example:

LOCK HELS, D2

The CATALOG of this diskette’s contents will now show an asterisk ( * ) next to the entry for HELS.

If you decide you no longer wish to keep the file LOCKed, the UNLOCK command will (surprisingly) unlock the file.

Example:

UNLOCK HELS

The syntax for the commands is

LOCK E [,Es] [,[Da] [,[P]

UNLOCK E [,[Es] [,[Da] [,[P]

The interpretation of the notation is discussed in the Syntax section of Chapter 3.

If you try to DELETE or RENAME a file that’s LOCKed you’ll receive the message FILE LOCKED You’ll also see this message if you try to SAVE a file using the name of a LOCKed file (if the file you’re trying to SAVE is in the same language as the LOCKed file).

If you try to SAVE a file using the name of a LOCKed file in a different language, then you’ll receive the message FILE TYPE MISMATCH Try again, using a different file name.

VERIFY

Occasionally information may not be recorded correctly on a diskette. This may happen if the diskette is scratched or dirty, for example. The VERIFY command reports a file which may be damaged or written incorrectly.
The syntax is the usual one for DOS commands:

VERIFY [\.\$] [.\$] [.\$]

Examples of the way to use the command follow:
VERIFY SAM
VERIFY FINANCE-8.D2.V22

VERIFY checks to see that information in the specified file is self-consistent. If it is, you see no message; the prompt character for the language you're using is simply printed:
> for Integer BASIC
> for AppleSoft
* for the Monitor.

However, VERIFY doesn't check to see whether or not a program is clobbered. If you saved a program that was messed up somehow, it still will be messed up on the diskette, and it will still VERIFY.

If the VERIFY command finds an error, the
I/O ERROR
message is displayed.

If you try to VERIFY a file that isn't on the disk, the message
FILE NOT FOUND
is presented.

You can use VERIFY from Integer BASIC, AppleSoft, or the Monitor. From these languages you can VERIFY any type of file, including text files (see Chapters 6, 7 and 8) and machine language programs (see Chapter 9).

WRITE-PROTECTING A DISK

The LOCK command allows you to protect a particular file. But sometimes you will want to be sure that all files on a certain diskette are not accidentally written over, and thus lost. To "write-protect" a diskette, you merely need to cover up the squarish write-protect cutout in the side of the disk. Stick-on adhesive labels are supplied for this purpose when you purchase diskettes but, in a pinch, any piece of sturdy tape will do.

Note that the System Master diskette does not have a write-protect cutout; it is permanently write-protected.

If you decide you want to rewrite a write-protected diskette, simply remove the label (often called a "tab") that covers the write-protect cutout.

Some programs cannot be used with a write-protected diskette. An example of such a program is ANIMALS, one of the demonstration programs of the System Master disk. Put your System Master in your drive, and boot DOS if you need to. Now type
LOAD ANIMALS
which will put the program into memory. Now type
RUN
and the message
WRITE PROTECTED
STOPPED AT 10:40
will be displayed. ANIMALS won't RUN on a write-protected diskette because it saves information on the diskette each time you play the game. When you RUN the program, the diskette in the drive must not be write-protected, else the information can't be written on the diskette.

Now ANIMALS is in memory, but you can't RUN it with the System Master diskette. Put an initialized diskette, one that is not write-protected, in the drive. Now type
RUN
and now you can play ANIMALS, a game that will "remember" what you "teach" it by saving the information on the diskette. When you're through playing, type
SAVE ANIMALS
so that you'll have the game on a diskette that's not write-protected.

If you type
CATALOG
you should see that you have not only a copy of ANIMALS on the diskette, but also a new file called ANIMALSFILE that was created by the program ANIMALS.

PROTECTING YOURSELF AGAINST DISASTER

Floppy disks are sturdy and reliable compared to some other ways of storing computer programs -- for example, on the backs of old envelopes. But it's still possible to lose or destroy all information on a diskette. A diskette may get scratched or damaged by heat; it may get lost, or a dog may chew it; someone may decide to use it as a frisbee at the beach; if a diskette isn't write-protected, it may accidentally get written over. And a diskette will eventually wear out -- a lifetime of 40 working hours is about average.

** Moral **

Keep more than one copy of a program around if you don't want to lose it. In computerese, "back up" any valuable program.

If you are in the midst of writing or modifying a program, one way to back up the program is to keep copies of earlier versions. Then if the current version is lost you can fall back to the next-most-recent version, and hopefully not lose too much programming time. One good way to do this is
to end each file name with a number which changes from version to version. For example, suppose you start to write a program called FINANCE. The first time you save the program, call it FINANCE-1. Next time you work on the program, save it under the name FINANCE-2; the third time, it becomes FINANCE-3, and so on. You will wind up with a whole collection of FINANCE programs, with the largest version number representing the most recent version of the program.

It's a good idea to SAVE a developing program periodically (with a new version number). If you do this every 15 or 20 minutes, an unexpected power failure or other disaster will not erase all your work. You can, of course, immediately continue working after saving the current state of the program -- just be sure to assign a new version number for the next SAVE. If the diskette starts filling up, DELETE some of the earlier versions. But it's a good idea to keep several versions around, in case something calamitous happens to the current version. Or you may just happen to want an earlier version -- not all revisions are improvements.

The phrase "backing up" is also used to describe keeping multiple copies of programs on separate diskettes. There are two approaches to backing up in this fashion. The first method, using one disk drive, is to simply SAVE the program on one diskette, remove that diskette from the drive, insert another diskette and SAVE the program again.

The second approach involves duplicating all the information on one diskette onto a second diskette. Details of the second approach are discussed in the next section.

**USING THE COPY PROGRAM**

You can use the COPY program, on the System Master diskette, to copy the entire contents of your current programming diskette onto your back-up diskette. If you are using DOS from AppleSoft Basic, COPY is the program to use for copying a diskette.

In these copying programs, the diskette from which copying is done is called the "original" diskette. The entire contents of the original diskette will be copied onto a "duplicate" diskette. The duplicate diskette does not have to be INITIALIZED before being copied onto. In fact, any previous information that was on the duplicate diskette will be erased.

Before copying the original diskette, it's a good idea to write-protect it. Then you can't accidentally erase its contents, even if you put it into the wrong drive.

As default values, the program initially assumes that the original diskette will be placed in the currently selected drive (the drive from which you ran COPY or COPYA) connected to the controller card in the currently selected slot. The duplicate diskette is assumed to be in the other drive connected to the same controller card (even if there isn't another drive there). To use any of these default slot or drive numbers, just press the RETURN key when the default number is shown. If any of the default numbers are wrong, either for the original or for the duplicate diskette, you must type the correct number when you see that default number on the screen.

Here's an example of using the COPY program on an Apple with one disk drive. It assumes your disk drive is attached to a disk controller card in the same slot as the one from which you ran COPY.

1) Place the System Master diskette in the currently selected drive. Type RUN COPY (RUN COPYA from AppleSoft) and after the usual whirring you should see

```
APPLE DISKETTE Duplication Program

ORIGINAL SLOT: DEFAULT = 6

1) To use the program's default slot number, slot #6 in this example, just press the RETURN key.

3) When the message

DRIVE: DEFAULT = 1

appears, press RETURN.

4) Press RETURN again when you see

DUPPLICATE SLOT: DEFAULT = 6

5) When you see the default drive number for the duplicate diskette:

DRIVE: DEFAULT = 2

6) The message

-- PRESS 'RETURN' KEY TO BEGIN COPY --

will appear. This is your signal to remove the System Master diskette from the drive and insert the original diskette, from which you wish to copy. (Did you remember to write-protect your original?)

7) Now the program will proceed, first by directing you to insert your original diskette, then by informing you that it is reading information from the original diskette. Then the program will let you know when it is ready for you to replace the original diskette with the duplicate diskette in the drive, and when it is formatting (INITIALIZING) that diskette. After the duplicate diskette is INITIALIZED you will be instructed to re-insert the original diskette so that the program can read some more information from it, and then to re-insert the duplicate
8) When copying is complete you’ll see the message

DO YOU WISH TO MAKE ANOTHER COPY?

If you type Y (yes), the copying procedure will be repeated, assuming
the same location(s) for the original and duplicate diskettes. Be sure
to re-insert the original diskette when you see

~~ PRESS ‘RETURN’ KEY TO BEGIN COPY ~~

and that you use another diskette (either a blank or one with unwanted
information on it) as the duplicate. If don’t wish to make another
copy, type N. You will leave the program and DOS will assume your
default disk drive to be the one which was the location of the duplicate
diskette.

If you have more than one drive, specify the slot and drive numbers for
the original and duplicate diskettes and insert the diskettes in these
locations before the copying begins. Note: If you have more than one disk
drive, the system won’t prompt you to remove and replace diskettes. It
only does that for single drive copying.

If you try to copy onto a write-protected diskette you’ll see the message
I/O ERROR
STOPPED AT (you’ll see a line number here)
Until the write-protect label is removed from the duplicate diskette no
information can be put onto the diskette.

An I/O ERROR or UNABLE TO READ message will also appear when the drive
door is open or if there is no diskette in the drive. These messages may
also indicate a problem with the diskette in the drive. Be aware that the
default drive number may have been changed after you have gotten one of
these messages. When you RUN the copy program again, you may have to
specify the drive location of the System Master diskette.
DEBUGGING: MON AND NOMON

The process of trying to get a program to run the way you want it to is called "debugging." Program errors are often referred to as "bugs." All disk commands and all information sent between the computer and the disk are normally not displayed on the screen. But when you're debugging, monitoring this information can help you track down problems.

The MON command allows you to monitor a variety of information. To turn various parts of the display on and off, again, use the NOMON (No Monitor) command.

Three different parameters may be used in these commands:
- C stands for commands to the disk (such as OPEN, READ, etc)
- I stands for input from the disk (when reading a file)
- O stands for output to the disk (when writing a file).

These parameters are used only with the NOMON and MON commands. Usually NOMON C, I, O is in effect: no monitoring is taking place.

The syntax for the commands is:
- MON [C] [I] [O]
- NOMON [C] [I] [O]

At least one of the three parameters must be present with the NOMON and MON commands, else the command will be ignored. The parameters may appear in any order and, as usual, must be separated by commas.

There are 7 different ways in which the MON command may be used:

<table>
<thead>
<tr>
<th>MON C</th>
<th>Commands to the disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON I</td>
<td>Input from the disk</td>
</tr>
<tr>
<td>MON O</td>
<td>Output to the disk</td>
</tr>
<tr>
<td>MON I,O</td>
<td>Input from and output to the disk</td>
</tr>
<tr>
<td>MON C,I</td>
<td>Commands to and input from the disk</td>
</tr>
<tr>
<td>MON C,O</td>
<td>Commands to and output to the disk</td>
</tr>
<tr>
<td>MON C,I,O</td>
<td>Commands to, input from, and output to the disk</td>
</tr>
</tbody>
</table>

*** Note ***
A MON command remains in effect until a NOMON, INT, or PP (FIRMWARE ONLY) command is encountered or until you boot the system or do a restart (30K). or RESET

A neat trick: you can issue a MON command and later cancel it without affecting the screen format -- even the NOMON does not show on the screen.

Suppose you execute a MON command, say
MON C, I, O
To cancel the command without having it print on the screen, include
PRINT DS: "NOMON C, I, O"; VTAB R2(77); CALL -868
where DS, as usual, contains CTRL-D.

MAXFILES

DOS allows up to 16 files to be active (in use) at one time. DOS deals with several types of files in addition to the BASIC program files discussed so far. See Chapter 6 for a discussion of sequential text files, Chapter 8 for random-access text files, and Chapter 9 for the DOS commands used with binary (machine language) files.

The MAXFILES command specifies how many active files are permitted. When you boot DOS, the command
MAXFILES 3
is executed, which sets up the default condition: a maximum of 3 files may be executed simultaneously until another MAXFILES command is executed.

The command's syntax is
MAXFILES n
where n must be an integer from 1 to 16. Specifying a value outside this range will cause a SYNTAX ERROR message from either Applesoft or Integer BASIC; from the Monitor, a beep is the only indication that you've done something wrong.

For each file specified, MAXFILES sets aside 595 bytes of memory space called a file buffer. This additional memory space for each active file is used to help adjust for the fact that memory speed is far faster than disk access speed. When you retrieve information from a diskette, DOS brings in 256 characters at a time and puts them in the "input" part of the file buffer, then delivers to you whatever subset of these 256 characters your program requested. If you are sending information to a diskette, characters are stored in the "output" part of the file buffer until 256 characters have accumulated, then they're shipped to the diskette all at once.

If you retrieve information from a diskette, DOS brings in 256 characters at a time and puts them in the "input" part of the file buffer, then delivers to you whatever subset of these 256 characters your program requested. If you are sending information to a diskette, characters are stored in the "output" part of the file buffer until 256 characters have accumulated, then they're shipped to the diskette all at once.

Suppose you have MAXFILES 1 and one file is active. An attempt to perform a DOS command (such as CATALOG) will cause the message
NO BUFFERS AVAILABLE to be displayed.

When the system is booted, the number of active files (n) defaults to 3, so 1785 bytes of memory are reserved for 3 file buffers. Under most circumstances, you won't need more than 3 active files. If more files are required, type
MAXFILES n
(where n is the number of needed files) in immediate execution mode before loading and running a program.

In immediate execution mode, changing MAXFILES erases Integer BASIC programs and increases MAXFILES messes up Applesoft strings, since WINEM: Is moved down without moving the program or strings. To avoid the problem, reset MAXFILES before loading and running a program.
If MAXFILES is used within a program, it changes memory pointers, and a GOTO, GOSUB, or other instruction can get lost. If you must change MAXFILES from within an AppleSoft program, make the MAXFILES command the first statement in the program, before any string variables are declared. For example, `10 PRINT CHR$(4); "MAXFILES 5":`.

To use MAXFILES from within an Integer BASIC program, you need to create an EXEcl file, as discussed at the end of Chapter 6.

**TRACE**

The AppleSoft TRACE command is a useful debugging tool. But when TRACE is in effect, DOS commands inside AppleSoft programs don’t work because TRACE prints the line number before the DOS command. There’s a partial solution to the problem. You can insert a RETURN (that is, CHR$(13)) in the DS string `10 DS=CHR$(13)+CHR$(6)` and then most DOS commands will work properly even if TRACE is in effect -- see the exception below.

If TRACE is in effect and DOS tries to READ an INPUT from the diskette, your program will instead stop and wait for an INPUT from the keyboard, which limits the usefulness of TRACE in connection with DOS.

**USING THE MASTER CREATE PROGRAM**

As discussed in Chapter 2, INIT is used to create slave diskettes. In this section you’ll learn how to create master diskettes. The distinction between a slave and a master is not readily apparent: both come charmingly attired in black plastic (no, not leather) garb. It’s up to you to revise your greeting program and diskette label to remind yourself which is slave and which is master.

The System Master diskette contains a program called MASTER CREATE that can run on an Apple II with at least 16K of memory. The MASTER CREATE program does the following for you:

* Converts a slave diskette (whose DOS is memory-size dependent) into a master diskette (whose DOS is self-relocating so that memory is used efficiently on any size system).

* Gives the converted diskette a new greeting program name, the name DOS will attempt to RUN each time the diskette is booted.

The MASTER CREATE program must be used with a diskette that has already been INITIALIZED. It will not work with a diskette that is write-protected.

Here’s an example of how to upgrade the diskette INITIALIZED in Chapter 2 (the one with the ONE TO TEN program on it) to convert the slave diskette created by INIT into a master diskette. For convenience, that diskette will be referred to as diskette ONE in the discussion that follows.

Before using MASTER CREATE, do the following:

1) Insert the diskette you wish to convert — diskette ONE for this example — into the disk drive, and RUN the diskette’s greeting program — named HELLO on diskette ONE. The message displayed by a greeting program should include the version of DOS used to initialize the diskette, and its status as slave or master.

2) Change the appropriate lines of the greeting program to display the new information, "MASTER DISKETTE". Then SAVE this new version of the greeting program. If the diskette’s outside label requires a similar change, make that change now.

3) Note the name of the greeting program. If you wish the converted diskette to RUN this same program each time it is booted, just as it did before converting, you will give this greeting program name to the MASTER CREATE program, later on. If you’ve always wished your greeting program had some other name than its present one, RENAME the greeting program now. Later, you will give the new name to the MASTER CREATE program.

To use MASTER CREATE, do the following:

4) Put the System Master diskette in the drive, boot DOS, and from either BASIC type BRUN MASTER CREATE.

5) You should see the message

```
DOS 3.2 MASTER-CREATE UTILITY
COPYRIGHT 1983 BY APPLE COMPUTER INC. ALL RIGHTS RESERVED
(NOW LOADING DOS IMAGE)
```

6) You’ll then be told to type the greeting program name to be used by the converted diskette:

```
PLEASE INPUT THE GREETING PROGRAM'S FILE NAME
```

44
We'll assume that when you saved the revised greeting program on diskette ONE (step 3, above) you used the name HELLO. So type HELLO unless you wish the diskette to run some other program name each time the diskette is booted. When you press the RETURN key to enter the greeting program name, you'll see this message:

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: HELLO. PLACE THE DISKETTE TO BE MASTRED IN THE DISK DRIVE. PRESS <RETURN> WHEN READY.

NOTE: IF YOU WANT A DIFFERENT FILE NAME, PRESS ESC.

7) Follow the instructions. Remove the System Master diskette from the disk drive and replace it with the diskette you wish to convert -- diskette ONE in this case. Finally, press the RETURN key to begin converting; the program will inform you when the process is complete.

8) After using the MASTER CREATE program, always re-boot DOS before doing any other work.

*** Note ***

The greeting program name that you give to the MASTER CREATE program is not placed in the diskette’s CATALOG. It just tells the diskette’s DOS which program name to run each time the diskette is booted. You must make sure that the diskette’s CATALOG actually contains a program bearing the name name you give to the MASTER CREATE program.

If you forget to do so (by skipping step 3 above), you’ll see the message FILE NOT FOUND each time you booted the disk using this diskette.

*** Reminder ***

You must remember the name of the greeting program for each diskette. Using the same greeting program name on all of your diskettes makes this easier.
TEXT FILES: AN INTRODUCTION

Sometimes you’ll want to use the disk to store information that is not a program. You may, for example, wish to keep copies of correspondence, a list of words used in a word-guessing game, intermediate results of a calculation, or a mailing list. A text file, sometimes called a data file, will allow you to do this and more. The latter I marks text files in the CATALOG directory.

Text files are created and retrieved using DOS commands in an Integer BASIC or AppleSoft program. A text file may be created using a program written in one language and retrieved from a diskette using a program written in another language.

Most sample programs in this manual are in AppleSoft. If you wish to convert the programs to Integer BASIC, recall that in Integer BASIC you can’t make string arrays and you must DIMension string variables. In an Integer BASIC command such as INPUT A$ , B$ , C$ only RETURNs (not commas) may separate the three responses. This manual does not tell you how to make each program run in Integer BASIC; see the Appendix M of the AppleSoft II BASIC Programming Reference Manual for details of converting between languages. For some hints about changing the BASIC in which a program runs, after the program has been written, see page 76 of this DOS manual.

The DOS commands LOAD and RUN (also BLOAD and BRUN) may not be used with a text file. An attempt to do so will cause the message FILE TYPE MISMATCH to appear. LOAD and RUN expect a BASIC program file (and BLOAD and BRUN expect a Binary machine-language file), not a text file. Instead, you must write programs that send data to a text file and retrieve data from a text file, using the DOS commands discussed in this chapter:
OPEN
CLOSE
READ
WRITE
APPEND
POSITION
EXEC

The commands OPEN, READ, WRITE, APPEND and POSITION cannot be used in immediate-execution mode. If you try to do so, you’ll receive the message NOT DIRECT COMMAND. These commands must be used in deferred-execution mode, that is, from within a program. The commands CLOSE and EXEC may be used in immediate-execution mode.

In addition to the commands listed above, the DOS commands LOCK and UNLOCK, DELETE, RENAME, NON and HOMOD, VERIFY, CATALOG work with text files in the same way they work with program files.

There are two different types of text files: sequential text files and random-access text files. Both types of text files store strings of ASCII codes to represent the data, but in different formats. Diagrams of the two text-file types are shown below (the character n represents the RETURN character, not automatically at the end of most PRINT statements).

"PICTURE" OF A SEQUENTIAL TEXT FILE

<table>
<thead>
<tr>
<th>Character:</th>
<th>7</th>
<th>1</th>
<th>A</th>
<th>T</th>
<th>O</th>
<th>N</th>
<th>B</th>
<th>L</th>
<th>O</th>
<th>W</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII:</td>
<td>55</td>
<td>13</td>
<td>65</td>
<td>84</td>
<td>13</td>
<td>79</td>
<td>78</td>
<td>59</td>
<td>13</td>
<td>66</td>
<td>76</td>
</tr>
<tr>
<td>File Byte:</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Field:</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"PICTURE" OF A RANDOM-ACCESS TEXT FILE

(Example: Record Length 5, One Field per Record)

<table>
<thead>
<tr>
<th>Character:</th>
<th>7</th>
<th>1</th>
<th>A</th>
<th>T</th>
<th>O</th>
<th>N</th>
<th>B</th>
<th>L</th>
<th>O</th>
<th>W</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII:</td>
<td>55</td>
<td>13</td>
<td>66</td>
<td>00</td>
<td>00</td>
<td>65</td>
<td>84</td>
<td>13</td>
<td>00</td>
<td>00</td>
<td>79</td>
</tr>
<tr>
<td>File Byte:</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Record Byte:</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Field:</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record:</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The terms "field" and "record" will be discussed in Chapters 6, 7 and 8. The commands OPEN, CLOSE, READ, WRITE and POSITION are used with both types of files, but in somewhat different ways. Sequential text files are simpler to use and understand, in some respects, so we will discuss the use and structure of sequential text files first. The use of random-access text files is described in Chapter 8. More detailed and technical information about all types of files can be found in Appendix C.

SEQUENTIAL TEXT FILES: SOME EXAMPLES

Suppose you want to make a file containing a list of words to be used in a word-guessing game. Here are two pairs of programs that deal with such a file. The first program in each pair creates a text file on the diskette. The second program in each pair retrieves the data stored in the text file from the diskette.

This program creates a text file named WORDS1, containing the words APPLe, BANANA, CATALOG, DORMANT, EAGLE, FRUIT, GOOSE, HAT and ICICLES.
As this point you'll have a file called WORDS1 on your diskette. WORDS1 will be marked with a "Y" in the CATALOG to indicate that it's a text file. The file consists of items of data (in this case, words) separated by RETURNS. A RETURN character is automatically sent at the end of every PRINT statement which does not end with a comma or a semicolon. Note that in this usage each RETURN is a character rather than an action — in particular, it is the character with ASCII code 13.

Each item of data, ending with its RETURN character, is called a field. A field is stored in the text file as a series of characters represented by their ASCII codes. The last character in each field must be the RETURN character, ASCII code 13.

WORDS1 is called a sequential text file because each field is stored beginning immediately after the RETURN character of the preceding field. When stored on the diskette, fields may be of different lengths; the word APPLE takes 6 bytes (one for each letter plus one for the RETURN character), BANANA takes 7 bytes, and so on. A sequential text file is stored on the diskette as one long, continuous series of ASCII-coded characters, a chain of fields with no gaps left between them.

Once WORDS1 is on the diskette, the question immediately arises, "How can I retrieve it?" The following AppleSoft program will retrieve WORDS1:

```
10 REM RETRIEVE WORDS1
20 D$ = "": REM CTRL-D
30 PRINT D$: "OPEN WORDS1"
40 PRINT D$: "READ WORDS1"
50 FOR I = 1 TO 2
60 READ X$1(I)
70 NEXT I
80 PRINT D$: "CLOSE WORDS1"
90 END
```

Line 30 OPENS the file; line 40 tells DOS that all subsequent INPUT or GET statements will refer to the named diskette file instead of the Apple's keyboard. It is as if the disk were typing responses, instead of you. An INPUT command always causes one complete field, ending with its RETURN character, to be "typed in" to the Apple. If another INPUT command follows, it will cause the next field to be read in, and so on. So lines 50 through 70 cause DOS to start at the beginning of WORDS1 and retrieve 2 fields which are placed in the array AS(1), AS(2), AS(3), AS(4), ... AS(9). line 80 politely CLOSES the file.

If MON C, I, 0 is not in effect when the above program is RUN, you will see nothing on your screen. But if MON C, I, 0 (or just MON C, I) is in effect, you'll see:

```
OPEN WORDS1
READ WORDS1
APPLE
BANANA
```
Commas in a disk-lens PRINT command usually send characters to defined tab-fields on the screen. However, commas do not serve this same formatting function in PRINT commands used when writing to the disk; these commas are treated as if they were semi-colons. In PRINTing to the disk, items separated by commas will be concatenated, with no intervening spaces inserted. A comma at the end of a PRINT command has the same effect as a semi-colon: no automatic final RETURN character is sent.

When the program MAKE WORDS2 is run with MON C, T, D in effect, you'll see

```
OPEN WORDS2
WRITE WORDS2
APPLE, BANANA, CATALOG, DORMANT, EAGLE,
FRUIT, GOOSE, MAT, ICICLE
CLOSE WORDS2
```

This Applesoft program retrieves WORDS2:

10 REM RETRIEVE WORDS2
20 DO = "", REM CTRL-D
30 PRINT DO: "OPEN WORDS2"
40 PRINT DO: "WRITE WORDS2"
50 PRINT "APPLE, BANANA, CATALOG, DORMANT, EAGLE,"
60 PRINT "FRUIT, GOOSE, MAT, ICICLE"
70 PRINT DO: "CLOSE WORDS2"
80 END

When the above program is run with MON C, T, D in effect, you'll see

```
OPEN WORDS2
READ WORDS2
APPLE, BANANA, CATALOG, DORMANT, EAGLE,
FRUIT, GOOSE, MAT, ICICLE
CLOSE WORDS2
```

In Integer BASIC, commas can separate multiple INPUT responses for numeric variables, but not for string variables. Only RETURN characters can separate multiple responses when INPUT is used with multiple string variables. In Integer BASIC, therefore, the program RETRIEVE WORDS2 will assign the entire field (9 words, 8 commas and 4 spaces) to the variable A$5. Then you will get the END OF DATA message when there is no field to assign A$5.

In Applesoft BASIC, you can also use the GET command to retrieve data from a text file, character by character. This has the advantage that you can define any character as marking the end-of-word, for instance. The following Applesoft program also retrieves the text file WORDS2.

In line 19, the CLEAR command sets all variables (including I and all A$5's) to zero. Line 29 uses Applesoft’s alternate way of setting D0 to CTRL-D (4 in the ASCII code for CTRL-D). This method avoids the invisible (and un-copyable) control character.

Note that the PRINT command in line 50 ends with a semi-colon. A semi-colon at the end of a PRINT command stops the automatic printing of a RETURN character after the last data character. Therefore the characters sent to the disk by the next PRINT command will appear in the same field with the characters sent by line 50’s PRINT command. The PRINT command in line 60 also ends with a semi-colon, so the field still does not have its end-marking RETURN character. Line 70’s PRINT command ends without a semi-colon or commas, allowing the automatic final RETURN character to be sent at last. This ends the field, which now contains all the characters PRINTed by lines 50, 60 and 70.
Line 80 GETs one character at a time from the text file WORDS2, which was opened for reading in lines 50 and 60. If the new character is neither a comma nor a RETURN, line 110 adds the new character to the end of the string AS(1). Then line 120 sends the program back to line 80, to GET the next character. Thus, the program builds up the first word, character by character, in AS(1).

When a comma is found the first word is ended, so line 50 sends the program back to line 70 to increment I and start collecting a new word in AS(2). And so on. Finally, a RETURN character ($8$) marks the end of the field, so line 100 sends the program on to line 130 to CLOSE the file and end the program. Note the use of CHR$(13)$, in line 80. You cannot directly type a RETURN character into a BASIC program line (a typed RETURN ends a program line), but CHR$(13)$ is a RETURN character in Applesoft.

When GET obtains characters from the disk, those characters are not displayed on the screen, even in MON C, I, O mode. Line 115 has been added to let you see the words as they are built up, character by character.

After an Applesoft GET command takes its response from a diskette text file, the following problems arise:

1. With ROMIN C, I, O the first character PRINTed after the GET will not appear on the screen.
2. With MON C, I, O the first character PRINTed after the GET will appear on the screen.
3. In either mode, if a DOS command is the first item PRINTed after the GET, the DOS command may not be executed because the necessary preceding RETURN is missing.

In the program GET WORDS2, the non-printing "throw-away" character CTRL-A ($7f$) was placed before the first desired PRINT character in line 115. This takes care of problems 1 and 2, above. To care problem 3, the RETURN character ($8$) was placed before the PRINTed DOS command in line 130, much as was done with TRACEx (see page 44).

When this program is RUN with MON C, I, O in effect, you will see the following (but all displayed in one column, not three):

The first field will contain APPLE, BANANA
and is 13 bytes long, one per character (commas must be counted too) plus one for the RETURN character. The second field, CATALOG, DORMANT, EAGLE is 22 bytes long; the third field, FRUIT, GOOSE, BAT, ICICLE is 23 bytes long.
When RUN with MON C, I, 0 in effect you'll see

(NEW WORDS3)
WRITE WORDS3
PRINT B: BANANAB
CATALOG: DORMANT, EAGLE
FRUIT: GOOSE, HAT, ICICLE
CLOSE WORDS3

Here's a program to retrieve WORDS3:

10 REM RETRIEVE WORDS3:A
20 D$ = CHR$(4) REM CTRL-D
30 PRINT D$: "OPEN WORDS3"
40 PRINT D$: "READ WORDS3"
50 INPUT R$: S$
60 INPUT T$: U$: V$
70 INPUT W$: X$: Y$: Z$
80 PRINT D$: "CLOSE WORDS3"
90 END

When RUN with MON C, I, 0 in effect, you'll see the following:

OPEN WORDS2
READ WORDS2
?APPLE, BANANA
?CATALOG: DORMANT, EAGLE
?FRUIT: GOOSE, HAT, ICICLE
CLOSE WORDS3

The program to READ the sequential text files WORDS1, WORDS2, WORDS3 were carefully designed to READ exactly the correct number of fields and the correct number of items per field. In general, a program to retrieve a text file must be designed around the specific file. If you make a mistake, the results can appear somewhat confusing. For instance, consider the following "wrong" program to retrieve the words in text file WORDS3.

10 REM RETRIEVE WORDS3:D
20 D$ = "" REM CTRL-D
30 PRINT D$: "OPEN WORDS3"
40 PRINT D$: "READ WORDS3"
50 INPUT R$: S$
60 INPUT T$: U$: V$
70 INPUT W$: X$: Y$: Z$
80 PRINT D$: "CLOSE WORDS3"
90 END

With MON C, I, 0 in effect, here's what you would see on RUNing the program.

OPEN WORDS3
READ WORDS3
?APPLE: BANANA
?CATALOG: DORMANT, EAGLE
?FRUIT: GOOSE, HAT, ICICLE
?EXTRA IGNORED
CLOSE WORDS3

The INPUT command in line 79 caused the entire field containing FRUIT, GOOSE, HAT, ICICLE to be READ into the Apple. The first three words were assigned to the variables W$, X$ and Y$. But there is no variable corresponding to the fourth INPUT response, ICICLE, so the message EXTRA IGNORED is displayed, and execution continues.

Here is another "wrong" program to READ the text file WORDS3:

10 REM RETRIEVE WORDS3:C
20 D$ = " " REM CTRL-D
30 PRINT D$: "OPEN WORDS3"
40 PRINT D$: "READ WORDS3"
50 INPUT R$: S$
60 INPUT T$: U$: V$: W$
70 INPUT X$: Y$: Z$
80 PRINT D$: "CLOSE WORDS3"
90 END

And here is a MON C, I, 0 RUN of the program.

OPEN WORDS3
READ WORDS3
?APPLE: BANANA
?CATALOG: DORMANT, EAGLE
?FRUIT: GOOSE, HAT, ICICLE
?EXTRA IGNORED
?
END OF DATA
BREAK IN 70

This time, line 58 caused the field
CATALOG: DORMANT, EAGLE
to be READ into the Apple. The three words are assigned to the variables T$, U$ and V$. But line 60's INPUT command expected four responses, so it causes the next complete field to be READ into the Apple: FRUIT, GOOSE, HAT, ICICLE.
The first word, FRUIT, is assigned to line 60's last variable, V$. There are no more variables with this INPUT command, so the message EXTRA IGNORED
is displayed, and execution continues. There are no more fields in the
file, so line 70’s INPUT command causes the
END OF DATA
message, and the program comes to a stop.

A somewhat more general pair of programs, MAKE TEXT and RETRIEVE TEXT are
discussed in a later section. They illustrate how to make a program more
adaptable to different text files.

OPEN-ING AND CLOSE-ING SEQUENTIAL FILES

Sequential text files should be used when information is to be retrieved
in a linear fashion from the beginning to the end of the file, and when
information does not require much updating or on-going revision.
For example, a sequential file could be used to contain data for a
word-guessing game, as in the preceding sample programs.

To create a sequential text file, the commands
OPEN
WRITE
PRINT
CLOSE
are used, in the order shown (though not necessarily right after each
other). To retrieve a sequential text file, the commands
OPEN
READ
INPUT
CLOSE
are used, again in the order shown though not necessarily right after each
other. Both procedures are illustrated in the preceding section.

A certain ritual is required before and after you create (WRITE) a
sequential text file: before using the file you must OPEN it. When you’re
done, you must CLOSE it. The same is true when retrieving (READING) a
sequential text file: OPEN before READING, and CLOSE the file when you’re
done.

STOP

Files that have been OPENed must be CLOSED. Failure to CLOSE a file that
was OPENed and written to by a WRITE command may result in loss of data.

The syntax for these commands is similar to other DOS commands.
[Note: OPEN and CLOSE are also used with random-access files -- see
Chapter 6.]

OPEN f [,So] [,Dd] [,Vv]
CLOSE [f]

Examples: OPEN SESAME
OPEN SHOP, D2, S7
CLOSE
CLOSE MOUTHED
CLOSE WINDOW

OPEN sets aside workspace in the Apple for the file f (for those who know
about such stuff, OPEN allocates a 39-byte file buffer to handle this
file’s input and output), and gets the system ready to read or write from
the beginning of the file. OPEN also sets up the slot and drive numbers
to be used by the subsequent WRITE (or READ) command.

The CLOSE command releases the workspace in the Apple (de-allocates the
file buffer associated with the file f). If f is not specified, all OPEN
files will be closed, with the exception of any file being used by the
EXEC command. EXEC files are discussed later in Chapter 7. OPEN
sometimes CLOSEs too: OPEN first checks to see if the named file is
already OPEN; if so, it CLOSEs it before re-OPENing it.

Note that the CLOSE command has no Drive or Slot parameters. If you type
CLOSE MYFILE
then any file named MYFILE will be CLOSED, regardless of the slot and
drive number associated with the file. Similarly, the command
CLOSE
will CLOSE all files (except a file being EXECed) on all disk drives.
In various circumstances, you may wish to delete a file f that may or may
not exist. This is especially important to avoid problems of overwriting
an old file (unless you overwrite the entire old file, part of the old
file will remain, attached to the end of your new file). Suppose a game
creator and uses the file SCORES each time it is played, and you wish your
program to delete any old file by that name at the start of each game.
The command
DELETE SCORES
will cause the error message
FILE NOT FOUND
if the file doesn’t exist, and your program will halt. Here’s a quick way
to delete any file named SCORES and re-OPEN it for new data, whether or
not that file already exists:

6 REM SCORES DELETE
10 D$ = "", REM D$ IS CTRL-D
15 PRINT D$: "OPEN SCORES"
20 PRINT D$: "DELETE SCORES"
25 PRINT D$: "OPEN SCORES"
30 REM
REMAINDER OF PROGRAM
HERE

WRITE-ING SEQUENTIAL FILES

Here is another program which creates a sequential text file. This
AppleSoft program creates a text file named SAMPLE which contains 3
strings and 18 numbers.

The file SAMPLE may or may not already exist each time the program is RUN;
if it does exist, it should be DELETED so as to remove old data from the
file. If it does not exist and your program tries to DELETE it, you’ll
receive the message
FILE NOT FOUND
and the program will stop. Lines 20 and 30 take care of the problem. If SAMPLE already exists, line 20 OPENs it and line 30 DELETEs it. If SAMPLE does not exist, line 20 creates a file SAMPLE and line 30 DELETEs it. When line 40 is executed it creates a clean new file SAMPLE, so the problem of mixed files is avoided.

5 REM MAKE SAMPLE
10 DO = CHR$(4): REM CTRL-D
20 PRINT DO: "OPEN SAMPLE"
30 PRINT DO: "DELETE SAMPLE"
40 PRINT DO: "OPEN SAMPLE"
50 PRINT DO: "WRITE SAMPLE"
60 PRINT "HI HI": PRINT "HI HI"
70 PRINT "OFF TO THE DISK WE GO"
90 FOR J = 1 TO 10
100 PRINT J
110 NEXT J
120 PRINT DO: "CLOSE SAMPLE"
130 END

Here's what you see on the screen when you RUN this program, if MON C, 1, 0 is in effect.

OPEN SAMPLE
DELETE SAMPLE
OPEN SAMPLE
WRITE SAMPLE
HI HI
HI HI
OFF TO THE DISK WE GO
1
2
3
4
5
6
7
8
9
10
CLOSE SAMPLE

Before you WRITE a file, it must be OPENed; CLOSE it (quietly, please) when you're done. Both the OPEN and WRITE commands must refer to the same file name.

Once a WRITE command is executed, my subsequent PRINT commands send all characters to the diskette, instead of the screen. A WRITE command is cancelled by the use of any DOS command in a PRINT statement. Even the "empty" DOS command (just CTRL-D) will do.

An INPUT command of the form INPUT X0 also cancels a WRITE command, but only after storing on the last text file character the ? which the INPUT command normally displays on the screen. If the form INPUT "WHAT'S YOUR NAME? "; X0 is used, the WRITE is canceled after the characters in the string are sent to the diskette.

An error message cancels a WRITE command, but only after the entire error message is stored as the last field in your text file.

The syntax for the WRITE command when used with sequential files is:
WRITE f
[Note: WRITE is also used with random-access files, see Chapter 8.]

Examples: WRITE LETTER
WRITE RIGHT

The sample program given at the beginning of this section is a simple illustration of the most basic (BASIC!) elements needed to create a text file. A slightly more general Apple-II program called MAKE TEXT is on the System Master diskette that came with your disk drive.

MAKE TEXT allows you to create a sequential text file containing up to 100 strings; each string may have at most 200 characters. Try it -- you'll like it (we hope). Place the System Master diskette in your drive and type
LOAD MAKE TEXT

A LISTING of the program should look like this:

5 REM MAKE TEXT
10 DIM AT(1000): I = 0
20 DO = CHR$(4): REM CTRL-D
30 HOME: TEXT
40 PRINT "YOU GET TO TYPE ONE STRING AT A TIME."
50 PRINT "A STRING MAY HAVE UP TO 200 CHARACTERS."
60 PRINT "THIS PROGRAM LETS YOU WRITE TEXT FILES."
70 PRINT "PICK ONE TO QUIT."
80 PRINT "PRESS RETURN KEY TO QUIT."
90 PRINT "TYPE STRING #"; I: I = I + 1

(Continued on next page)
(Continued from previous page)

...A PRINT "I = I + 1" /* CLOSE */
诩 190 PRINT I - L
lox FOR J = L TO I - 1
cox PRINT A(J)
ixo NEXT J
ixo PRINT 84, "CLOSE ", N$.

Once the program is Loaded, SAVE it on a diskette that's not write-
protected. (This step is necessary because this program, like the ANIMALS-
program discussed in Chapter 4, creates a new file.)

Is MAKE TEXT still in your Apple? And a non-write-protected diskette in the
drive? If so, type

MOD C, I, 0

so you can see the commands sent to and from the disk. Then type run
and you should see the following message:

YOU GET TO TYPE ONE STRING AT A TIME.
A STRING MAY HAVE UP TO 255 CHARACTERS.
THIS PROGRAM LETS YOU WRITE TEXT FILES.
(PRESS THE RETURN KEY TO QUIT.)
TYPE STRING #1:

Type in as many strings as you like (up to 100 may be entered). Warning:
the program uses INPUT, so don't type commas or colons into your strings.
When you wish to quit, just press the RETURN key instead of typing a
string. You'll be asked

WHAT FILE NAME?

Choose a name for your text file, press the RETURN key, and as your strings
are sent to the disk you'll see them printed on the screen. First will
appear the disk commands

OPEN f
DELETE f
OPEN f
WRITE f

(where the f is replaced by the file name you chose). They'll be followed by
a number ---- the number of strings you entered into the file. (This
number will be used by a program discussed in the next section that
retrieves your file.) Next you'll see your strings. Finally you'll see
the message
CLOSE f.

Here's a sample RUN of the MAKE TEXT program:

THIS PROGRAM LETS YOU WRITE TEXT FILES.
YOU GET TO TYPE ONE STRING AT A TIME.
A STRING MAY HAVE UP TO 255 CHARACTERS.

(TYPE STRING #1: HERE'S STRING 1)

(TYPE STRING #2: AND MY SECOND STRING)

(TYPE STRING #3: ON UP TO)

(TYPE STRING #4: ENOUGH ALREADY)

(TYPE STRING #5: WHAT FILE NAME? TEST
OPEN TEST
DELETE TEST
OPEN TEST
WRITE TEST
4
HERE'S STRING 1
AND MY SECOND STRING
ON ME GO
ENOUGH ALREADY!
CLOSE TEST

STOP

If you OPEN a text file that already exists and then WRITE to it (without
first DELETING the file and re-OPENING it), then you will overwrite at
least a portion of the file. Unless you overwrite at least as many
characters as existed in the old file, the result is that the new file
contents will be a mix of the data PRINTed to the file on the two
occasions. First will appear the new characters you PRINTed to the file
this time, and then will follow any portion of the old file you did not overwrite. To clear all characters from the old file, OPEN and DELETE the old file before you OPEN it anew. (In the program MAKE TEXT, lines 140 and 150 take care of “cleaning out” any previous text file by the same name.) To keep programs from overwriting a file, LOCK the file.

**READ-ING SEQUENTIAL FILES**

The DOS command READ allows you to retrieve a text file. Once a READ is executed, any subsequent INPUT statements (or GETs in Applesoft) refer to the specified file instead of the Apple's keyboard. This Applesoft program retrieves the text file SAMPLE created by the program listed at the beginning of the preceding section. READ, like WRITE, must be preceded by OPENing the file to be used. The file must be CLOSED as well.

```
5 REM RETRIEVE SAMPLE
10 DO = CHR$(4): REM CHR$(4)
     IS CTRL-D
20 PRINT DO: “OPEN SAMPLE”
30 PRINT DO: “READ SAMPLE”
40 INPUT N, R, C#
60 FOR I = 1 TO 10
60 INPUT W
70 NEXT I
80 PRINT DO: “CLOSE SAMPLE”
```

An OPEN must precede a READ, and an INPUT (or, in Applesoft, a GET) must follow a READ. The OPEN and READ must refer to the same file name. If you NOW the program with MOD C, T, O in effect you’ll see this:

```
OPEN SAMPLE
READ SAMPLE
*H1 NO
*?H1 NO
*OFF TO THE DISK WE GO
*7
*6
*5
*4
*3
*2
*1
CLOSE SAMPLE
```

The program was written explicitly with the SAMPLE file in mind; it assumes that the text file contains 3 strings, (A5, B5, and C5 in line 40) and 10 integers (W in line 60). Two question marks are printed when B5 and C5 are INPUT because RETURNs separated the INPUT’s multiple responses.

A READ command is cancelled by the use of any DOS command in a PRINT statement. The “empty” DOS command (just CTRL-D) will do just fine. Use of the PR, or IN# commands also cancels a READ.

The syntax for the READ command is the same as for WRITE:

```
READ f
```

[Note: READ is also used with random-access files, see Chapter 8.]

**Examples:** READ LETTER

```
READ CAREFULLY
```

Stopping a READ in Applesoft using CTRL-C will generate a string of REINTERS. To avoid this, press the RESET key to stop the program.

An Applesoft program that retrieves text files created by the MAKE TEXT program is on the System Master diskette. Place the System Master diskette in your drive and type

```
LOAD RETRIEVE TEXT
```

then SAVE the program on the same diskette you used for MAKE TEXT. (The program is really a companion piece to MAKE TEXT, and it’s simply more convenient to have them on the same diskette.)

A LISTing of the program should appear as follows:

```
10 DO = CHR$(4): REM CHR$(4)
12 PRINT “THIS PROGRAM RETRIEVES TEXT FILES”
14 PRINT “CREATED BY THE ‘CREATE TEXT’ PROGRAM.”
16 PRINT “MON C,1,0 IS IN EFFECT.”
18 PRINT
20 INPUT “NAME OF TEXT FILE? ”, Z
22 PRINT DO: “MON C,1,0”
24 PRINT
30 PRINT DO: “OPEN “,Z
40 PRINT DO: “READ “,Z
50 INPUT I
55 DIM ?(1)
60 FOR J = 1 TO I
70 , INPUT ?(J)
70 NEXT J
90 PRINT DO: “CLOSE “,Z
100 PRINT DO: “UNMON C,1,0”
```

64

65
Now type
RUN
and you should see the message

THIS PROGRAM RETRIEVES TEXT FILES
CREATED BY THE "MAKE TEXT" PROGRAM.
MON C:10 IS IN EFFECT.

NAME OF TEXT FILE?

Type in the name of the text file you created using the MAKE TEXT program, press the RETURN key, and you should be off and running (oops -- rather, READING).

Here's what you'll see if the file TEST, used as a sample at the end of the last section, is retrieved using the RETRIEVE TEXT program:

THIS PROGRAM RETRIEVES TEXT FILES
CREATED BY THE "MAKE TEXT" PROGRAM.
MON C:10 IS IN EFFECT.

NAME OF TEXT FILE? TEST

OPEN TEST
READ TEST
"4"
"HERE'S STRING 2"
"AND MY SECOND STRING"
"HOW DO"
"ENOUGH ALREADY!"
CLOSE TEST
MON D:10

MORE ON SEQUENTIAL FILES: APPEND AND POSITION

The DOS commands APPEND and POSITION, respectively, allow you to add text to the end of a sequential text file, and to access information from any specified field within a text file.

APPEND allows you to add data to the end of a sequential text file. This is particularly useful if you wish to extend the information in a sequential text file, as in the ANIMALS program discussed in Chapter 4 could have. The command OPEN, you will recall, always sets the position-in-the-file pointer to byte 0, the first character in the file. The command APPEND performs an OPEN for you on a file that already exists, then sets the position-in-the-file pointer to one byte beyond the last character in the file.

The following program builds a file called TESTER that contains the two strings "TEST 0" and "TEST 1":

9 REM MAKE TESTER
10 D4 = CHR$(47) REM CTRL-D
20 PRINT D4: "OPEN TESTER"
30 PRINT D4: "APPEND TESTER"
40 PRINT D4: "WRITE TESTER"
50 PRINT D4: "DELETE TESTER"
60 PRINT D4: "CLOSE TESTER"
70 PRINT D4: "TEST 0"
80 PRINT D4: "TEST 1"

The following program APPENDs the strings "TEST 2", "TEST 3" and "TEST 4" to the file TESTER:

5 REM APPEND TEST
10 D4 = CHR$(47) REM CTRL-D
20 PRINT D4: "APPEND TESTER"
30 PRINT D4: "WRITE TESTER"
40 PRINT D4: "TEST 2"
50 PRINT D4: "TEST 3"
60 PRINT D4: "TEST 4"
70 PRINT D4: "CLOSE TESTER"

The following program displays the file TESTER:

5 REM RETRIEVE TESTER
10 D4 = CHR$(47) REM CTRL-D
20 PRINT D4: "OPEN TESTER"
30 PRINT D4: "READ TESTER"
40 PRINT D4: "READ TESTER"
50 FOR I = 1 TO 5
60 INPUT A$ 
70 NEXT I
80 PRINT D4: "CLOSE TESTER"

APPEND must be followed by WRITE (attempting to READ will just cause the END OF DATA message). The syntax for the APPEND command is doubtless familiar if you've been reading straight through this manual:

APPEND f [sa] [v4] [v6]

APPEND, even though it is used only for WRITErings into a text file, does not cause the FILE LOCKED message if the file is locked. That message is given only if you attempt to actually WRITE to the file.

The DOS command POSITION allows you to WRITE or READ information beginning in any given field of a sequential text file. The syntax for the POSITION command is

POSITION f [R4]

where R4 is the Relative-field position. This command specifies that DOS's position-in-the-file pointer will be moved forward (only) to the p-th field ahead of the current pointer position. If p=0, the following READ or WRITE begins in the current field. If p=1, the following READ or WRITE skips the current field and begins in the next field. If p=2, the
following READ or WRITE skips two fields including the current field, before beginning to READ or WRITE. And so on. If your file does not contain any field corresponding to the relative-field position specified by the POSITION command, the message END OF DATA will be displayed, and program execution will stop.

POSITION with the Bp parameter specifies a relative field position, p fields ahead of the current field. POSITION must refer to a file that you have already OPENed. OPEN automatically sets the position-in-the-file pointer back to the beginning of the first field. Thus, if POSITION is used immediately after OPEN, the relative-field position also corresponds to the actual, or absolute, field position. In no other case is this true.

Like any other DOS command, POSITION cancels a READ or a WRITE. Therefore, POSITION must be used before the associated READ or WRITE.

POSITION actually scans the contents of the file, byte by byte, looking for the Bp-th RETURN character. If, during this process, it encounters an "empty" (value 0) byte, the message END OF DATA is presented immediately. It is not necessary to actually INPUT or GET any such null character.

Here is a program that uses POSITION to retrieve various fields from the TESTER file, created earlier by the MAKE TESTER and APPEND TESTER programs:

```
010 REM POSITION TESTER
20 REM = CHR$(4). REM CTRL-O
20 PRINT D8: "OPEN TESTER"
40 PRINT D8: "POSITION TESTER, R2"
50 PRINT D8: "READ TESTER"
60 INPUT A8
70 PRINT D8: "POSITION TESTER, R1"
80 PRINT D8: "READ TESTER"
90 INPUT B8
100 PRINT D8: "OPEN TESTER"
110 PRINT D8: "POSITION TESTER, R3"
120 PRINT D8: "READ TESTER"
130 INPUT C8
140 INPUT D8
150 PRINT D8: "CLOSE TESTER"
```

If you RUN this program with NOE C, I, D in effect, you will see:

```
OPEN TESTER
POSITION TESTER, R2
READ TESTER
?TEST 2
POSITION TESTER, R1
```

Are you surprised at the results? Remember that the current field is relative-field position number 0. Also, remember that each INPUT causes one field to be READ into the Apple, and advances the position-in-the-file pointer to the beginning of the next field.

BYTE-ING OFF MORE

Note: the following section is not for beginners, and sequential files can be used perfectly well without a knowledge of the parameters discussed here.

The DOS commands WRITE and READ can be used with a Byte parameter to WRITE or READ information starting from any place in a text file -- if you know where that place is. The trick involves knowing at exactly which byte in the file you want to start (each byte contains one character's ASCII code). To do this, you must know exactly how you have stored information into the file. You must count all RETURNS, commas, spaces and other characters in the file when figuring out where to begin. The problem is even more difficult for WRITE, because you must also know where to end.

The B parameter is an actual or absolute position in the file unless R is specified. If R is given, the B parameter is the actual position within the specified field.

The command WRITE Command, B27
sets the position-in-the-file pointer to the twenty-eighth byte of the file named THESMTH (the first byte is number 1). Characters sent to the disk by a subsequent PRINT command will replace an equal number of characters that already existed in the file, beginning with the character in the 28th byte.

This over-writing is not confined to the current field. If you PRINT fewer than the number of characters remaining in the current field, you will create two new fields: the field you just PRINTed, followed by the tail-end of the field you were over-writing. If you PRINT more than the number of characters remaining in the current field, you will over-write some of the characters at the start of the next field; the current field will then be longer, and the next field shorter than before.
It is also possible to WRITE into bytes that are beyond the last byte of an existing sequential text file. An attempt to READ the intervening un-written bytes will cause the END OF DATA message to be displayed, and your program will stop. See the discussion of READ with the B parameter, for information on accessing sequential text file fields that are not next to each other.

The syntax for this command is

```
WRITE F [,...],b
```

where the B parameter specifies the file byte at which characters sent by the next PRINT command will begin replacing file characters. The default value of B is 0, the first byte in a file. Byte b is an absolute, position within the file. The B parameter may specify a position either before or after the current position-in-the-file pointer. [Note: this command is also used with random-access files. See Chapter 6.]

Similarly, the command

```
READ LASTMONTH, B32
```

sets the position-in-the-file pointer to the thirty-third byte of the file named LASTMONTH (again, the first byte is number 0). A subsequent READ command will cause all characters in the next field (i.e., up to the next RETURN character), beginning with the character whose ASCII code is stored in the file's 33rd byte, to be READ into the Apple. If the 33rd byte does not contain the first character of a field, only the remaining characters in that field will be READ.

```
Syntax for this command is
```

```
READ F [,...],b
```

where the B parameter specifies the file byte where the next INPUT or GET command will begin reading characters. The default value of B is 0, the first byte in a file. Byte b is an absolute, position within the file. The B parameter may specify a position either before or after the current position-in-the-file pointer. [Note: this command is also used with random-access files, see Chapter 6.]

The following program sets the position-in-the-file pointer to byte 14 (the fifteenth byte) in file TESTER, which was created earlier by the program MAKE TESTER. Then it WRITEs the string "APPLE COMPUTER". Note the familiar sequence: OPEN, then WRITE and PRINT, and finally CLOSE.

```
5 REM BYTE READER
10 D4 = CHR$(4): REM CTRL-D
20 PRINT D4: "OPEN TESTER"
30 PRINT D4: "READ TESTER.B14"
40 INPUT A#
50 PRINT D4: "CLOSE TESTER"
```

Try to predict what you will see, before you RUN this program.

The following program sets a pointer to byte 18 in the file TESTER, just modified by the preceding program. Then this program READs to the next RETURN in the file. Again the familiar format: OPEN is followed by READ, next come INPUT statements (or, in AppleSoft, GETs may be used) and finally the file is CLOSED.

```
5 REM BYTE READER
10 D4 = CHR$(4): REM CTRL-D
20 PRINT D4: "OPEN TESTER"
30 PRINT D4: "READ TESTER.B18"
40 INPUT A#
50 PRINT D4: "CLOSE TESTER"
```

With MON C, I, O in effect, RUN RETRIEVE TESTER to see how the previous program has changed the file TESTER. As you can see, the field containing APPLE COMPUTER has completely over-written the fields that contained TEST.
CHAPTER 7

AUTO APPLE

74 Controlling the Apple via a Text File: EXEC
73 Creating an EXEC File
76 Capturing Programs in a Text File
77 Converting Machine-Language Routines to BASIC
78 MAXFILES and Integer BASIC Programs
78 EXECutive Session
To better understand the contents of this chapter, it is suggested that you first read Chapter 6, on sequential text files.

**CONTROLLING THE APPLE VIA A TEXT FILE: EXEC**

The DOS command EXEC is similar to RUN, except that the disk file used by an EXEC command is a text file that contains commands or program lines, including BASIC statements, as if they were typed at the keyboard.

To initiate a demonstration of some EXEC command abilities, LOAD EXEC DEMO from your System Master diskette and then SAVE it on a diskette that's not write-protected. Leave the un-write-protected diskette in the drive, since the program WRITEs a text file.

Next RUN the program. You should see the message

```plaintext
<< EXEC DEMO >>
THIS PROGRAM CREATES A SEQUENTIAL TEXT FILE NAMED "DO'ER" CONTAINING SEVERAL
STRINGS. EACH A LEGAL APPLE II COMMAND. WHEN 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.
```

Press the Apple's space bar, and after a brief pause you should see the disk drive's IN USE light come on as the program writes the DO'ER file onto the diskette. Now type EXEC DO'ER press the RETURN key. Your Apple will begin a solo performance based on the script in the DO'ER file.

Here's a brief summary of the major things DO'ER does:

First DO'ER issues a MON C.1.0 command, so you'll be able to see what happens.

Second, a three-line program is written and saved on diskette under the name NEW PROGRAM!. The program is then LISTED.

Next a FOR-NEXT loop is executed to take up some time, so you get a chance to look at the screen before the activity continues.

Next DO'ER uses the INT command to enter Integer BASIC, loads the program COLOR DEMOS, and LISTs it.

At this point, DO'ER uses CALL -155 to enter the Monitor and executes some machine-language instructions before using the FP command to enter Applesoft.

From Applesoft a MON C.1.0 command is executed, then NEW PROGRAM! is RUN, modified, LISTed (again a FOR loop allows you to take a look at the screen) and SAVED using the name EVEN MORE RECENT PROGRAM!

Lastly, the program NEW PROGRAM! is DELETED and the CATALOG (including the new addition EVEN MORE RECENT PROGRAM!) is displayed.

And you won't even have to lay a finger on the keyboard (unless your CATALOG has more than 18 entries, in which case you need to press the space bar to see the balance of the CATALOG entries).

**CREATING AN EXEC FILE**

Here's a step by step example to illustrate how to create an EXEC file named DOIT that contains the following commands:

```
LIST 20, 5#
RUN AWAY
CATALOG
```

First create and SAVE an Applesoft program called AWAY to use in the above demonstration:

```
5 REM AWAY
10 PRINT "AWAY AWAY WITH RUN BY G.M."
```

Next write and SAVE the following program, called MAKE EXEC, that will create a text file called DOIT. When you later EXEC DOIT, the commands your MAKE EXEC program has PRINTed into the DOIT text file will tell Apple to RUN the AWAY program for you. Notice that the commands which are PRINTed into the DOIT file, for later EXECing, are not preceded by a CTRL-D.

```
5 REM MAKE EXEC
10 DO$ = CHR$(4): REM CHR$(4) IS CTRL-D
20 PRINT DO$: "OPEN DOIT"
30 PRINT DO$: "WRITE DOIT"
40 PRINT "LIST 2D-DO"
50 PRINT "RUN AWAY"
60 PRINT "CATALOG"
70 PRINT DO$: "CLOSE DOIT"
```

After you have MAKE EXEC and AWAY both SAVED on a diskette, type the command

```
RUN MAKE EXEC
```
to create a sequential text file named DOIT.
Type the command
EXEC DO IT

to cause the commands in the file DO IT to be executed one by one, just as if they'd been typed in from the keyboard. Again, notice that the commands new being EXECuted were not preceded by a CTRL-D in the program

MAKE EXEC. First lines 20 through 50 from the program currently in memory (probably the program MAKE EXEC) are listed. Then the program named
"AWAY" is RUN, and finally the CATALOG on the diskette is displayed.

CAPTURING PROGRAMS IN A TEXT FILE

Here's a far more useful example of using the EXEC command: it allows you to capture program listings as text files. Such a program can be used for

* translating Integer BASIC programs into Applesoft
* renumbering parts of programs and EXECing them anywhere into another program
* inserting favorite subroutines into programs from a subroutine file on the diskette by EXECing the file
* "appending" one program to another
* repair programs that have become partially unreadable
(you can capture the good portion in a text file, re-boot, then EXEC the program portion back into memory)

The line numbers 2240 and 5140, following the LIST command in line 6 of the CAPTURE program, should be replaced by the line numbers of the program in memory that you wish to capture. The name of the sequential text file containing the listing is LISTING.

1 REM CAPTURE
2 DB = CHR$(4) : REM CTRL-D
3 PRINT DB : "OPEN LISTING"
4 PRINT DB : "WRITE LISTING"
5 POKE 33,30
6 LIST 2240,5140
7 PRINT DB : "CLOSE LISTING"
8 TEXT : END

We made the line numbers of this program very close together, so that you can add these lines to a program already in memory, or anywhere within your program that you have eight free line numbers. You could just as easily put all the lines of CAPTURE above the highest numbered line in your program.

CAPTURE creates a text file containing commands that are preceded by line numbers. When you EXEC that text file, the numbered commands will not be executed. Instead, just as if you had typed those lines in from the keyboard, the lines are stored as a program in Apple's memory. Once captured in a text file, a program can be modified and then EXECed back into Apple's memory. Unlike LOAD or RUN, EXEC does not delete a program that is already in memory. Using CAPTURE, you can capture a program in a text file from one language, then EXEC the program back into another language (of course, the program may not run without some changes-there's somewhat different syntax for Integer BASIC and Applesoft). You can also use EXEC this way to add new lines to an existing program in memory. In fact, you can save a listing of CAPTURE in a text file named LIST SAVEX, say, and then EXEC LIST SAVEX any time you wanted to add the CAPTURE program to a program in memory.

CONVERTING MACHINE-LANGUAGE ROUTINES TO BASIC

Here's another useful program that will take a machine-language routine and convert it into a BASIC program portion which POKEs the machine-language routine into memory. The program portion can be used as part of either an Applesoft or an Integer BASIC program, to put the machine-language routine into memory each time the BASIC program is run.

05 REM CODE-POKES WRITER
10 DS = "" : REM CTRL-D
15 PRINT DS : "OPEN CODE-POKES"
20 PRINT DS : "DELETE CODE-POKES"
25 PRINT DS : "OPEN CODE-POKES"
30 PRINT DS : "WRITE CODE-POKES"
35 LINENUMBER = 7800
40 FOR PLACE = 750 TO 790
45 COUNTER = COUNTER + 1
50 IF COUNTER = 10 THEN COUNTER = 1
55 IF COUNTER < 1 THEN 120
60 PRINT
65 PRINT LINENUMBER:
70 PRINT LINENUMBER = LINENUMBER + 1
75 PRINT " POKE ",PLACE": " : PEER
80 PRINT " (PLACE), " : ":"
85 NEXT PLACE
90 PRINT
95 PRINT
100 PRINT DS : "CLOSE CODE-POKES"
105 END

When you use this program, the number in line 40 should be changed to contain the line number of your BASIC program where the POKEing program portion is to start. The FOR loop in line 50 should contain the starting and ending decimal memory locations of the machine-language routine you wish to convert.

Once you've typed in the program, running it will create the text file CODE-POKES. Now use the command
EXEC CODE-POKES
to place your machine-language-POKEing program portion into any other program, beginning at the line number previously specified. The program
CODE-POKES WRITER will work with either Applesoft or Integer BASIC.
MAXFILES AND INTEGER BASIC PROGRAMS

An EXEC file must be used if you want to increase MAXFILES from inside an
Integer BASIC program without erasing your program. Here's how. Use the
procedures described above to create an EXEC file, let's call it FILE.EX.

FILE.EX should contain the following commands to allow for 5 files on a
system:

```
MAXFILES 5
LOAD PROGRAM
DL 10, 20
RUN
```

The first lines of the program would be as follows; note that what appears
as CTRL-D is created by holding down the key marked CTRL while typing the
letter D.

```
10 PRINT "CTRL-D EXEC FILE.EX"
20 END
30 REM MAIN PROGRAM
```

EXEC-UTIVE SESSION

The usual syntax for the EXEC command is

```
EXEC f
```

where f is the name of a sequential text file containing BASIC commands or
program lines. Examples of this usage appear throughout the earlier
sections of this chapter. EXEC with this syntax causes the first field of
file f to be read into the Apple as if it were being typed on the
keyboard. When the first field's RETURN character is "typed", the Apple
attempts to execute the field's contents as a BASIC command, or enter the
field's contents as a BASIC program line. The type of BASIC (Integer or
Applesoft) is not changed by EXEC unless the file contains an FF or INT
command. When execution has ceased on the first field, the second field of
file f is read into the Apple and treated similarly. The action comes to a
stop when the last field of file f has been read.

The EXEC command cannot be stopped by CTRL-C.

Only one EXEC file can be OPEN at any one time. If you are EXECing a file,
and one of the commands thus executed is another EXEC command, the first
EXEC file is immediately CLOSED. Thereafter, it is the second EXEC
command that is being executed.

If a file being EXECed contains a command to RUN any program, EXEC will
wait patiently until the program ends. Then the next EXEC file command
will be executed.

However, if a program is RUNning while an EXEC file is OPEN, any INPUT
statement in the program will take the next field in the file being EXECed
as its response, ignoring the keyboard. Worse yet, if that response is an
immediate-execution DOS command, the command will be executed before the
program continues. Results can be very confusing.

If you interrupt a RUNning Applesoft program by typing CTRL-C while an
EXEC file is OPEN, the remainder of the EXEC file will usually not be
executed.

If any field of an EXEC file cannot be interpreted as a valid BASIC
command or program line, the message

```
SYNTAX ERROR
```

is generated, and the next field is read into the Apple. Thus, you can
EXEC any text file, whether or not it contains BASIC statements (first
be sure you've SAVED any program in memory). In MON C, 1, 0 mode, this
can provide a crude but handy tool for quickly examining the contents of a
text file.

The EXEC command can also be used with the Relative-field position
parameter, in a way that is a bit different from POSITION's use of that
parameter. The syntax for this use is

```
EXEC f [,Rp]
```

where Rp specifies that file f is to be EXEC'd starting in the p-th field
of file f. Since EXEC always sets the position-in-the-file pointer to the
first character of the file, so the parameter Rp always indicates the p-th
field relative to the file's beginning. Thus p always corresponds to the
file's actual, or absolute, field. Rp indicates that EXECing begins
with the file's first field, R1 indicates EXECing begins with the second
field, etc.

Note that this is different from POSITION's use of the R parameter,
where R1 is a relative field only, and may indicate different actual
file fields at different times in a program.

EXEC HTFILE, RF
generates an
END OF DATA
message if the R parameter specifies the second field beyond the file's
end. (If the first field beyond the file's end is specified, nothing
happens).
CHAPTER 8
USING RANDOM-ACCESS FILES

82 Random-Access Files: How They Work
82 A Specific Record
84 Multiple Records
86 A Demonstration: The RANDOM Program
88 WRITING and READING Random-Access Files
For a better understanding of the information presented in this chapter, it is suggested that you first read Chapter 6, on Sequential files.

**RANDOM-ACCESS FILES: HOW THEY WORK**

Random-access text files are like a collection of equal-sized cells in a honeycomb — some cells may be full, others may be empty. Each "cell" is called a record. When you create a random-access file, you must specify the standard size for the records the file is to contain.

Unlike the fields in sequential files, which may be of almost any length, the records in a random-access file are of specified fixed length. The first time you write to any particular record in a file, enough space is set aside on the diskette for a complete, standard-length record, whether or not the record is actually filled. So random-access files don't necessarily represent an efficient use of space. However, since these files are set up in such a regular fashion, it's fast and easy to retrieve or modify information from any part of the file — hence the name "random-access" file.

Random-access files should be used in applications requiring fast access to various parts of the file, or where individual pieces of information in the file need to be changed fairly often. For example, a random-access file is particularly suitable for maintaining a mailing list.

Random-access files are created and retrieved in a manner very similar to that used for sequential files. The main difference is that certain commands have additional parameters: OPEN requires an L (Length of record) parameter, while READ and WRITE each use an R (Record number) parameter. Some sample programs will be presented and discussed before getting into details on creating and retrieving random-access files and how the new parameters work. More technical information about random-access text files may be found in Appendix C.

**A SPECIFIC RECORD**

How can you access a specific record in a random-access file? The following pair of Applesoft programs illustrates how DOS allows you to do this. The first program requests a name ($A), a telephone number ($F) and a zip code ($Z), then sends them to record 1 of a file called MAILER:

```
10 REM MAKE MAILER
20 D$ = CHR$(4); REM CTRL-D
30 INPUT "NAME: ";$A
40 INPUT "PHONE: ";$F
50 INPUT "ZIP CODE: ";$Z
60 PRINT D$; "OPEN MAILER, L200"
70 PRINT D$; "WRITE MAILER,R1"
80 PRINT D$; PRINT $A; PRINT $F; PRINT $Z
90 PRINT D$; "CLOSE MAILER"
```

Line 20 places a CTRL-D in the variable $D, as usual.
Lines 30 through 50 request the information to be stored.
Do not type any commas or colons in your responses.
Line 60 OPENs a file called MAILER, with 200-byte long records.
Line 70 prepares for recording information in record 1.
Line 80 sends $A, $F and $Z to the diskette — since record 1 was specified in line 70, all three pieces of information are recorded in record 1, separated by RETURNs.
Line 90 CLOSEs the file.

**WITH MON C, I, O in affect, when the program is RUN you'll see:**

```
NAME: ANY DORKS
PHONE: (425) 555-1010
ZIP CODE: 92014
OPEN MAILER, L200
WRITE MAILER, R1
ANY DORKS
(425) 555-1010
92014
CLOSE MAILER
```

Record 1 of the file MAILER can be retrieved by this program:

```
10 REM RETRIEVE MAILER:
20 D$ = CHR$(4); REM CTRL-D
30 PRINT D$; "OPEN MAILER, L200"
40 PRINT D$; "READ MAILER, R1"
50 INPUT N1$; P1$; Z1$
70 PRINT D$; "CLOSE MAILER"
```

When RUN with MON C, I, O, you'll see the following. As usual, the pair of question marks indicates an INPUT with more than one response.

```
OPEN MAILER, L200
READ MAILER, R1
ANY DORKS
(425) 555-1010
92014
CLOSE MAILER
```

And here is a slightly different program to retrieve record 1 of MAILER.

```
10 REM RETRIEVE MAILER:
20 D$ = CHR$(4); REM QUOTES CONTAIN
   CTRL-D
30 PRINT D$; "OPEN MAILER, L200"
40 PRINT D$; "READ MAILER, R1"
50 INPUT N1#
```

82

83
MULTIPLE RECORDS

The program that created the random-access file MAILER wrote to a single record in the file, saving three different pieces of information separated by RETURNs. The next program demonstrates writing to several records. In particular, record numbers 12 through 15 of a random-access file called RA-FILE.

5 REM MAKE NEW RA-FILE
10 D$ = CHR$(4): REM CTRL-D
20 PRINT D$: "OPEN RA-FILE"
30 PRINT D$: "DELETE RA-FILE"
40 PRINT D$: "OPEN RA-FILE: R:"
50 FOR I = 12 TO 15
60 PRINT D$: "WRITE RA-FILE, R": I
70 PRINT "NAME ADDRESS ": I
80 NEXT I
90 PRINT D$: "WRITE RA-FILE, R13"
100 PRINT "DOS"
110 PRINT D$: "CLOSE RA-FILE"

Line 10 sets D$ to CTRL-D.
Lines 20 and 30 make sure RA-FILE is a new file.
Line 40 opens the file RA-FILE, whose records will each be 30 bytes in length.
Lines 50 through 59 create a loop that writes the information NAME ADDRESS followed by the record number, for records 12 through 15. Note that you must specify each record in a new WRITE command, before having PRINT send characters to that record.
Lines 90 and 100 change the information in record 13 to the text given in line 100’s PRINT command.
Line 110 closes the random-access file RA-FILE.

If 30 C, 1, 0 is in effect when the program is run, you’ll see the following:

OPEN RA-FILE
DELETE RA-FILE
OPEN RA-FILE: L30
WRITE RA-FILE, R12
NAME ADDRESS 12
WRITE RA-FILE, R13
NAME ADDRESS 13
WRITE RA-FILE, R14
NAME ADDRESS 14

WRITE RA-FILE: R13
NAME ADDRESS 15
WRITE RA-FILE, R13
DOS
CLOSE RA-FILE

In a similar fashion, you can read information from a selected record of a text file. The next program retrieves records 12 through 15 of the file called RA-FILE, trying, on line 58, to find which record(s) contains the letters "DOS" as the first three characters.

5 REM RETRIEVE RA-FILE
10 D$ = CHR$(4): REM CTRL-D
20 PRINT D$: "OPEN RA-FILE: L30"
30 FOR J = 12 TO 15
40 PRINT D$: "READ RA-FILE, R": J
50 INPUT A$
60 IF LEFT$(A$, 3) = "DOS" THEN PRINT "RECORD ": J
70 NEXT J
80 PRINT D$: "CLOSE RA-FILE"

Line 10 sets up CTRL-D in D$.
Line 20 opens the text file RA-FILE, whose records are 30-bytes long (that’s what we specified when the file was created in an earlier program, remember?).
Line 30 reads through 70 READ records 12 through 15 of RA-FILE.
Note that you must specify each record in a new READ command, before a subsequent INPUT will read characters from that record. In line 58, each record comes in from the disk as an ASCII string terminated by a RETURN. Line 60 checks the 3 leftmost characters of the INPUT string A$ from record r, to see if the word "DOS" is there. If it is, the message "RECORD r WAS CHANGED." is printed and the search continues.
Line 80 closes the file.

Here’s what you’ll see when you run the program, if 30 C, 1, 0 is in effect:

OPEN RA-FILE: L30
READ RA-FILE: R12
?NAME ADDRESS 12
READ RA-FILE: R13
DOS
RECORD 13 WAS CHANGED.
READ RA-FILE: R14
?NAME ADDRESS 14
READ RA-FILE: R15
?NAME ADDRESS 15
CLOSE RA-FILE
Notice that when the file was retrieved only records that had been written to were examined. If you had asked for record 8 in RA.FILE, you would have received the END OF DATA message, since no information had been written to that record of the file. Similarly, had you tried to INPUT more than one field from any of the existing records you would have been given the same message: each of records 12 through 15 contains only one field.

A DEMONSTRATION: THE RANDOM PROGRAM

Last but by no means least, the System Master diskette contains a program called RANDOM that uses a random-access text file to demonstrate a small inventory control scheme. And by small we mean small: the program can handle at most 9 parts. This keeps the program simple. The Apple, of course, is capable of handling thousands of parts in an inventory.

First, the program copies itself and the random-access text file APPLE PROMS used to keep track of the inventory, then it automatically RUNS the program for you. 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. Remember to press the RETURN key each time you complete a response.

1) From the System Master, RUN RANDOM and you should see the message

    THIS DEMONSTRATION WILL NOT EXECUTE ON A WRITE-PROTECTED DISKETTE SUCH AS YOUR DOS SYSTEM MASTER FOR YOUR CONVENIENCE, PROVISIONS HAVE BEEN MADE TO COPY THIS PROGRAM AND IT'S DATA TO ANOTHER DISKETTE.

    DO YOU WISH TO DO THIS NOW? (Y OR N) Y

If you type N for "no" in response to the above message, you'll find yourself back in Applesoft.

2) Press Y FOR "yes". You'll see the message

    NOW READING DATA.

Followed by the message

    INSERT AN INITIALIZED DISKETTE; THEN PRESS THE RETURN KEY TO BEGIN TRANSFER.

3) Remove the System Master diskette, and place a non-write-protected diskette in the drive, then press the RETURN key. You'll perhaps catch a glimpse of the message

    WHEN THE PROGRAM AND DATA HAVE BEEN FULLY TRANSFERRED, THE PROGRAM WILL BEGIN RUNNING.

and then the program will begin execution.

4) Now you should see this:

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

Press 1 and you should see this message:

    PART NUMBER 1-9 (0=ALL) 0

5) Press 0, to get a list of all "parts" in this "inventory system" and you'll see

    PART# NAME          SIZE IN STOCK
    ----- ---------  --------
    1 PARALLEL PRINT 256 500
    2 COMMUNICATIONS 256 1230
    3 (NOT AVAILABLE) 256 0
    4 (NOT AVAILABLE) 256 0
    5 DISK BOOT 256 432
    6 STATE MACHINE 256 468
    7 SERIAL PRINTER1 256 876
    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 the RETURN key.

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

Choice 2 allows you to change any or all part names and descriptions. For example, suppose part 3 should be named COSMIC GUARD, size 56, with 1234 in stock. Here's how to revise the entry for part 3:

select option 2, CHANGE select part number 3
the old part name is displayed, with the cursor at its start, to allow you to enter the new name; when you
press the RETURN key the cursor will move to the right
and perform similarly for part size and quantity.

Just press the RETURN key by itself.

Choice 3 will stop the program.

**WRITE-ING AND READ-ING RANDOM-ACCESS TEXT FILES**

When used with random-access files, the CLOSE command works exactly as it does with sequential files (see "OPENing and CLOSEing Sequential Files" in Chapter 5). However, the syntax for OPEN has an additional parameter, the L parameter, which is required.

OPEN f [,j [,Sj] [,DJ] [,YV]]

The "L" stands for "length-of-record"; the number j indicates how many bytes (characters and digits) are to be allotted to each record in the random-access file you're creating (or, if you're retrieving a file, the number that were allotted when the file was created). If the L option is omitted, j is assigned the default value of 1. The number j must be in the range 1 through 32767.

When you OPEN a file prior to WRITEing, if you specify a different Length parameter, then you specified when you OPENed prior to WRITEing the file, DOS will blindly use the new Length parameter to calculate record positions within the file. You will have to keep detailed written documentation on the structure and contents of your files (some programmers keep such information in record #0 of the file). It's helpful to always include the Length parameter in each file's name, with such names as:

RANDFILES: L20
STOCKLISTS: L100
DIRECTORIES (L10)

There is no way to find the length of a record in a random-access file; you must make this information part of your documentation.

Records should never be longer than the number of bytes specified by the L parameter; records may be partially over-written or combined with confusing results.

**WRITE** and **READ** each have an R parameter, to be used when creating or retrieving particular records in random-access files.

**WRITE** f [, ,Rr]
**READ** f [, ,Rr]

Examples: WRITE LEGIBLY, R3,
READ FAST, R13

The R (Record) parameter is used to create (with WRITE) or retrieve (with READ) the rth record of the file. The default value of r is 0, specifying the first record of a file.

Using CTRL-C to stop a READ in AppleSoft causes a string of REENTS to be generated; press the RESET key instead.

In some respects, each separate record in a random-access text file may be treated as a short sequential file. WRITE and READ can be used with a Byte parameter in addition to their R parameter. The Byte parameter specifies the beginning byte of the specified record, for the next PRINT (after WRITE) or INPUT or GET (after READ).

**WRITE** f [, ,Rr] [, ,Bb]
**READ** f [, ,Rr] [, ,Bb]

If specified, the B (Byte) parameter causes WRITEing (or READING) to begin at the b-th byte of the specified record. The default value of b is 0, the first byte of a record. The B parameter may specify a position in the record either before or after the current position-in-the-file pointer. Using the B parameter necessitates a thorough, detailed, byte-by-byte knowledge of the contents of each record in the file.

Once READ or WRITE has moved the position-in-the-file pointer to a particular record, POSITION can also be used to move the pointer ahead (only) to further relative-field positions within the record. However, POSITION cancels either WRITE or READ mode (without changing the position-in-the-file pointer), so another WRITE or READ command (this time with no parameter) is necessary to re-instate that mode.

Details on how information is stored on the diskette in general, and in random-access files in particular, may be found in Appendix C.
CHAPTER 9
USING MACHINE LANGUAGE FILES

Machine Language Files
SAVE
LOAD
RUN
The MIS Subroutine
MACHINE LANGUAGE FILES

DOS allows you to store on diskette, and retrieve from diskette, the information in your Apple II's memory. You have already seen the DOS commands SAVE, LOAD and RUN; these commands deal with the contents of Apple's program memory, interpreted as commands in BASIC programs. The DOS commands discussed in this chapter -- BSAVE, BLOAD and BRUN -- perform similar functions, but they deal with the contents of any portion of Apple's memory, in its uninterpreted, raw binary-and-hexadecimal form.

The B before each of the following commands stands for a binary file; a B also precedes the name of binary files in the CATALOG. A binary file is just an exact, bit-for-bit copy of the information that was stored in a specified range of Apple memory locations. Those locations may have contained a machine-language program, binary data, or a bit-mapped "picture" from Apple's high-resolution graphics screen.

BSAVE

The BSAVE command creates a file named f and stores all the contents of a segment of memory. The syntax is

BSAVE f, A4, Lj, [S], [D], [V]

where as usual the S, D, and V parameters stand for slot number, drive number, and volume number. Note that the A and L parameters are not optional.

The A parameter specifies the starting address (in either decimal or hexadecimal code) of the memory portion to be stored on diskette. A dollar sign ($) must precede an address expressed in hexadecimal. If the A parameter is less than 0 or greater than 65535, a SYNTAX ERROR message is displayed. Therefore, equivalent negative addresses may not be used with this command. Within the range 0 through 65535, no error message is generated if the A parameter specifies a starting memory address that does not correspond to actual, installed memory chips. In practice, it is not useful to specify an A parameter greater than the maximum memory address in your Apple (49151 or $BFFF on a 48K system).

The L parameter specifies the length, in bytes, of the memory portion to be stored. If the L parameter is less than 0 or greater than 65535, a SYNTAX ERROR message is generated. If the L parameter is 0 or in the range 32768 through 65535, a RANGE ERROR message is generated. 32767 is the greatest number of bytes that can be stored in a single field on the diskette. If you wish to store more than 32767 memory locations, use two BSAVEs. Within the range 1 through 32767, no error message is generated if the L parameter specifies a range of memory addresses, not all of which correspond to actual, installed memory chips. In practice, it is not useful to specify a range of memory addresses extending beyond the maximum memory address in your Apple (49151 or $BFFF on a 48K system).

BLOAD

The BLOAD command returns the contents of a Binary file to your Apple II's memory. BLOAD does not erase a BASIC program in memory, unless the data is BLOADed into the particular portion of memory containing your program.

The syntax is

BLOAD f, [As], [S], [D], [V]

where the S, D, and V parameters are as usual. If the A parameter is used, then the Binary file's contents replace a portion of the existing contents of Apple's memory, beginning at address a. If the A parameter is not used, the file's contents are returned to the same Apple memory locations whose contents were originally BSAVED. See BSAVE for a complete discussion of the A parameter.

Assume the binary file PICTURE contains a high-resolution picture. Either of these examples places the picture into the first high-resolution graphics area of the Apple's memory:

BLOAD PICTURE, A8192
BLOAD PICTURE, A52000

Either example also clobbers the RAM version of Applesoft.

Note: a machine-language program may no longer be executable if it is moved to a memory location different than the one from which it was saved.

BRUN

The syntax of the BRUN command is the same as for BLOAD:

BRUN f, [As], [S], [D], [V]

The Binary file f should be a machine-language program.

First BRUN does a BLOAD. If the A parameter is given, the file's contents are placed into Apple's memory beginning at location a. If the A parameter is not used, the file's contents are returned to the same Apple memory locations whose contents were originally BSAVED. See BSAVE for a complete discussion of the A parameter.

After BLOADing the file, BRUN does a machine language jump (JMP) to location a. If the file was a machine-language program, this begins execution of that program.
THE RWTS SUBROUTINE

Normally, user access to and from the DISK II is restricted to the use of DOS. However, another method of accessing the DISK II is available to machine language programmers. You may skip this section if you're not familiar with machine language.

The DISK II can be accessed directly from machine language through the use of the RWTS subroutine, which is part of the DOS. The "RWTS" stands for "Read or Write a Track and Sector". In the following explanation, any numbers preceded by $ are hexadecimal numbers.

Every diskette initialized by the DISK II drive is separated into 35 tracks, numbered 0 to 34. These tracks may be thought of as grooves on a phonograph record, except that they are not connected with each other.

Basically, the tracks are arranged in separate concentric circles, with the large hole in the center of the diskette forming the common center of the circles. Track 0 is on the outer edge of the diskette, while track 34 is nearest the center. The disk drive has a "head" that acts very much like the needle on a record player, except that the head on the disk drive is magnetic. This head moves to different tracks on the diskette, where it either reads information off of the diskette, or writes information onto the diskette.

Each track on the diskette consists of 16 sectors. Sectors are pre-defined groupings on each track, that allow the user to work with single blocks of 256 bytes, rather than with the entire 35 sectors of bytes that fit on one track. The sectors within a track are individually numbered, consecutively, 0 to 15 around the diskette. As the diskette spins, each sector will pass underneath the head, at which time the head may write to or read from that sector. Each sector consists of two portions: the address field and the 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 diskette. The data field contains an encrypted form of the actual 256 bytes of data which were stored on that sector.

The "Read or Write a Track and Sector" subroutine (referred to as the RWTS subroutine), allows the user to write information to, or read information from, any track and sector on the diskette, via machine language. In order to use the RWTS subroutine, the user must first create an I0B (Input/Output control block) table, and an accompanying "Device Characteristics Table". The I0B tells the RWTS subroutine which slot and drive number the diskette will be in, which volume number to expect on the diskette, which track and sector to access, and whether to read from or write to the diskette. The Device Characteristics Table provides some information to the RWTS subroutine that is necessary to operate the Apple DISK II.

To use the RWTS subroutine, the user must set up the I0B and the Device Characteristics Table somewhere in memory. A "controlling subroutine" must be written to place the I0B into memory. The subroutine must JSR to the starting address of the RWTS subroutine (at location $290). When the RWTS subroutine is jumped to, the A and Y registers must contain the address of the starting location of the I0B. The A register must contain the high

address byte, and the Y register the low address byte. The format of the I0B is given in Table 1, at the end of this section. Table 4 gives the format of the Device Characteristics Table.

Here is an example of how to use the RWTS subroutine. The sample I0B, Device Characteristics Tables, and a controlling subroutine will all be loaded into memory just after location $2000.

The following controlling subroutine will load the A and Y registers with the address of the starting location of the I0B, and then jump to the RWTS subroutine.

```
$0000 A9 6C LDA #$6C Load A register with #$6C
$0003 A8 8A LDX #$8A Load Y register with #$8A
$0006 20 39 03 JSR #$39 Jump to RWTS subroutine
$0009 60 RTS Return from JSR
$000C 00 BRK Break
```

The following I0B is in one that you would use to access slot 6, drive 1, to write 256 bytes of memory starting at location $2000, onto track 18, sector 6 of the diskette:

<table>
<thead>
<tr>
<th>Location</th>
<th>Code</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0B00</td>
<td>@1</td>
<td>TOR typed index, must be $01</td>
</tr>
<tr>
<td>$0B02</td>
<td>6B</td>
<td>Slot number times 16</td>
</tr>
<tr>
<td>$0B03</td>
<td>01</td>
<td>Disk drive number</td>
</tr>
<tr>
<td>$0B04</td>
<td>00</td>
<td>Expected volume number</td>
</tr>
<tr>
<td>$0B05</td>
<td>12</td>
<td>Track number</td>
</tr>
<tr>
<td>$0B06</td>
<td>05</td>
<td>Sector number</td>
</tr>
<tr>
<td>$0B07</td>
<td>28</td>
<td>Low-order byte of Device Characteristics Table</td>
</tr>
<tr>
<td>$0B08</td>
<td>0C</td>
<td>High-order byte of Device Characteristics Table</td>
</tr>
<tr>
<td>$0B09</td>
<td>00</td>
<td>Low-order byte of data buffer starting address</td>
</tr>
<tr>
<td>$0B0A</td>
<td>20</td>
<td>High-order byte of data buffer starting address</td>
</tr>
<tr>
<td>$0B0B</td>
<td>00</td>
<td>Unused</td>
</tr>
<tr>
<td>$0B0C</td>
<td>02</td>
<td>Command code, $02 = write</td>
</tr>
<tr>
<td>$0B0D</td>
<td>00</td>
<td>Error Code</td>
</tr>
<tr>
<td>$0B0E</td>
<td>00</td>
<td>Actual volume number</td>
</tr>
<tr>
<td>$0B0F</td>
<td>6B</td>
<td>Previous slot number accessed</td>
</tr>
<tr>
<td>$0B10</td>
<td>01</td>
<td>Previous drive number accessed</td>
</tr>
</tbody>
</table>

The following Device Characteristics Table must be included; we'll place it at location $290, just after the I0B. Locations $2B0 and $2B1 in the I0B above point to the address of the Device Characteristics Table's starting location.

<table>
<thead>
<tr>
<th>Location</th>
<th>Code</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2B0</td>
<td>00</td>
<td>Device type code (put a $00 here)</td>
</tr>
<tr>
<td>$2B1</td>
<td>01</td>
<td>Number of phases per track (put a $01 here)</td>
</tr>
<tr>
<td>$2B2</td>
<td>EF</td>
<td>Time count (put a $EF here)</td>
</tr>
<tr>
<td>$2B3</td>
<td>0E</td>
<td>Time count (put a $0E here)</td>
</tr>
</tbody>
</table>
When you have loaded the 108 at $C0A$, the Device Characteristics Table at $SC26$, and the controlling subroutine to load the A and Y registers at $SC06$, then

```
CALL $3072
```

will cause the entire routine to execute.

<table>
<thead>
<tr>
<th>TABLE 3: FORMAT OF IOB</th>
<th>Byte#</th>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ISTYP</td>
<td>Tells the RMTS subroutine what type of IOB this is. Should be a $81. No other type codes are currently defined.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ISLOT</td>
<td>Must contain the number of the slot times 16, in which the disk drive's controller card is located. For example, if you want to access slot #5, the value $50$ must be stored in this location.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>IDSRT</td>
<td>Must contain the number of the disk drive to be accessed -- either $91$ or $92$.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ISRTN</td>
<td>The volume number of the diskette to be accessed must be stored here. Volume $50$ will match the volume number assigned to any diskette.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ITADDR</td>
<td>The number of the track (0 to 36) to be accessed is stored here. Must be within the range $60$ to $22$.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ISECT</td>
<td>The number of the sector (0 to 15) to be accessed is stored here. Must be within the range $60$ to $2F$.</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>TDCHAR</td>
<td>These two bytes must contain the address of the starting location of the Device Characteristics Table (see below). Byte 7 must contain the low-order byte of the address, and byte 8 must contain the high-order byte.</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>IBDUFF</td>
<td>Bytes 9 and 10 must contain the address of the starting location of the &quot;data buffer&quot;. The data buffer is a 256-byte long section of memory upon which the RMTS subroutine will operate. If you are writing onto the diskette, the information in the data buffer will be written onto the diskette. If you are reading from the diskette, the information that comes off of the disk will be stored in memory at the location of the data buffer. 256 bytes is both the minimum and the maximum amount of information that can be read or written by the RMTS subroutine.</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>IBMOD</td>
<td>In this byte is stored the command code that tells the RMTS subroutine exactly what to do. The values that can be stored in byte 13 are: $50$ -- Null command. Does nothing but start the disk drive and position the head. $91$ -- Read the entire 256 bytes stored on the diskette at the specified track and sector, and store them in memory at the location of the data buffer. $92$ -- Write the next 256 bytes stored in memory at the location of the data buffer on to the diskette at the specified track and sector. $A4$ -- Format the diskette. When a diskette is formatted, self-synchronizing nibbles are written on every track and sector on the diskette. Because all of the diskette is formatted, the values in bytes 5 and 6 are ignored. All of a formatted diskette is available for use; there is no DOS, or anything stored on the diskette until the user puts something there.</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>ISTAT</td>
<td>This location will contain the code number for any error that may be encountered during execution. If the RMTS subroutine returns with the carry flag clear, no error has occurred. If it returns with the carry flag set, this byte indicates what type of error has occurred. $10$ -- Diskette in write-protected, and cannot be written to. $20$ -- Volume mismatch error. The volume number of the diskette found was different than the volume specified in byte 4. $60$ -- Drive error. Something unusual is happening. $80$ -- Read error. The RMTS routine was, after 48 repeated attempts, unable to read either the address field or the data field. If the data field for the specified sector has never had anything written on it, then a read error will result, because there is nothing to read.</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>IBMODD</td>
<td>The volume number of the diskette that is actually found will be stored in this location.</td>
<td></td>
</tr>
</tbody>
</table>
### TABLE 3: FORMAT OF IOB (continued)

<table>
<thead>
<tr>
<th>Byte#</th>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>IOBP2N</td>
<td>This byte must contain the slot number times 16 of the slot that was accessed most recently. For example, if you previously accessed a disk drive in slot 3, store the value $03$ here. If there is no controller in the specified slot, the disk will hang.</td>
</tr>
<tr>
<td>17</td>
<td>IOBPDN</td>
<td>This byte must contain the number of the disk drive that was accessed most recently -- a $01$ or $02$.</td>
</tr>
</tbody>
</table>

### Table 4: FORMAT OF DEVICE CHARACTERISTICS TABLE

<table>
<thead>
<tr>
<th>Byte#</th>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DEVTC</td>
<td>Device type code, telling what type of device this is. A $00$ should be stored in this byte for use with a DISK II.</td>
</tr>
<tr>
<td>2</td>
<td>PFTC</td>
<td>A $01$ should be stored here.</td>
</tr>
<tr>
<td>344</td>
<td>MNTC</td>
<td>Motor on time count complemented, in 100 micro-second intervals. A $0F$ should be in byte 3, and a $08$ in byte 4, for use with a DISK II.</td>
</tr>
</tbody>
</table>
SELECTING I/O DEVICES: IN#, PR# AND OTHER STUFF

There are various ways in which information can be used as input or output for your Apple computer. Very often the keyboard serves as a source of input. Usually the Apple uses the TV screen for output, but any accessory or peripheral connected to a controller in one of the seven Apple accessory slots can be used for input or output using the IN# and PR# commands.

Examples:

IN# 6 obtains subsequent input from the device controlled from slot #6. Note: if slot #6 contains a disk controller card, this command will cause DOS to be booted. If no device is in slot #6, the system may "hang". Press the RESET key to recover.

IN# 0 obtains subsequent input from the keyboard (not slot #0), instead of a peripheral device.

PR# 1 transfers output to the device controlled from slot #1, usually the printer. Note: if no device controller card is installed in slot #1, the system may "hang" and you'll have to press the RESET key to recover.

PR# 0 returns output to the TV screen (not to slot #0).

The syntax for the commands is

IN# [n]
PR# [n]

where n specifies the slot to use. What happens depends on n:

<table>
<thead>
<tr>
<th>value of n</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>less than 0</td>
<td>SYNTAX ERROR</td>
</tr>
<tr>
<td>0</td>
<td>re-establishes usual device (for IN#, input from the keyboard; for PR#, output to the TV screen)</td>
</tr>
<tr>
<td>1 through 7</td>
<td>transfers to device controlled from the specified slot (boots DOS if a disk controller card is in that slot)</td>
</tr>
<tr>
<td>8 through 16</td>
<td>SYNTAX ERROR in deferred-execution mode; the system hangs in immediate-execution mode</td>
</tr>
<tr>
<td>17 through 65535</td>
<td>RANGE ERROR</td>
</tr>
<tr>
<td>greater than 65535</td>
<td>SYNTAX ERROR</td>
</tr>
</tbody>
</table>

The command IN# 0 re-establishes input from the keyboard; PR# 0 re-establishes output to the TV screen.

With DOS in effect, the IN# and PR# commands may be used in immediate execution mode in the usual way (see your BASIC manual). But when they are issued by lines in a program, IN# and PR# must take the form of DOS commands such as

IN# D5; 'IN# 5'
PR# D5; 'PR# 5'

When DOS is not in effect, the IN# and PR# commands set the contents of the Apple Monitor Input and Output registers to select the desired input and output devices.

When DOS is in effect, the contents of the Apple Monitor Input and Output registers are set to select DOS, while the contents of the DOS Input and Output registers are set to select the desired input and output devices. The following paragraphs describe what happens each time a character leaves or enters the Apple.

When the Apple sends an output character, the Apple Monitor Output register directs that character to DOS. If the character is to be sent on (because it is not part of a DOS command), DOS does a fast two-stage switch:

1. First, DOS replaces the contents of the Apple Monitor Input and Output registers with the contents of the DOS Input and Output registers. Then it sends the character out to the device now selected by the contents of the Apple Monitor Input and Output registers.

2. Next, DOS re-connects itself by again placing the pointers to DOS in the Apple Input and Output registers.

Similarly, each time the Apple asks for an input character, the Apple Monitor Input register directs that request to the DOS. Once again, DOS does its fast two-stage switch:

1. First, DOS replaces the contents of the Apple Monitor Input and Output registers with the contents of the DOS Input and Output registers. Then it obtains an input character from the device now selected by the Apple Monitor Input and Output registers.

2. Next, DOS re-connects itself by again placing the pointers to DOS in the Apple Input and Output registers.

When DOS is in effect, DOS intercepts all input characters from the input device before they reach Applesoft or Integer BASIC or the Monitor. That is why IN# and PR#, when typed on the keyboard as immediate-execution commands, can be trapped and used by DOS to change the DOS Input and Output registers.

Similarly, DOS intercepts all output characters from the Apple before they reach the output device (but after they have affected the Apple Monitor Input and Output registers). That is why IN# and PR#, if issued from within a program but not in PRINTed DOS commands, can disconnect DOS by changing the Apple Monitor Input and Output registers before the commands ever get to DOS. Because the entire contents of the Apple Monitor Input
and Output registers are replaced each time DOS attempts to send or receive a character, DOS will usually re-connect itself if it was not disconnected at both Input and Output registers simultaneously.

If you execute a PR# command from within a program, with a program line such as

50 PR# 1

then DOS will be partially disconnected and unable to intercept subsequent output. DOS is still connected for input, and the next attempt to obtain any input character will cause DOS to re-connect itself for both input and output.

The same situation occurs with the use of IN# inside programs when DOS is in effect. A program line such as

60 IN# 1

will disconnect DOS for subsequent input, DOS is still connected for output, and the next attempt to send an output (even a return or a prompt character) will cause DOS to re-connect itself for both input and output. To avoid such conflicts and allow DOS to manage the Input and Output registers, issue PR# and IN# commands in immediate-execution mode, or as DOS commands in program lines such as those mentioned earlier:

10 DS = ""; REM CTRL-D
20 PRINT DS; "PR# 1"
30 PRINT DS; "IN# 2"

The CTRL-D character tells DOS that the following output characters are a DOS command.

---

**TABLE 1: APPLE MONITOR INPUT AND OUTPUT REGISTERS**

**Monitor Input Register: Locations 56-57 ($38-$39)**

<table>
<thead>
<tr>
<th>When Register</th>
<th>To the</th>
<th>Then subsequent input comes from</th>
</tr>
</thead>
<tbody>
<tr>
<td>contents are set by</td>
<td>value</td>
<td></td>
</tr>
<tr>
<td>RESET</td>
<td>-741</td>
<td>Monitor Input Routine</td>
</tr>
<tr>
<td>@ CTRL-K [Note 1]</td>
<td>($FDB)</td>
<td>Monitor Input Routine from Apple keyboard</td>
</tr>
<tr>
<td>IN# [Note 2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>CTRL-K [Note 1]</td>
<td>49132 + s*256</td>
</tr>
<tr>
<td>IN#s [Note 2]</td>
<td>($C966)</td>
<td>If slot #s contains disk controller, then boot DOS</td>
</tr>
<tr>
<td>[ where s&gt;0 ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOS boot</td>
<td>-$574 + Top of mem.</td>
<td>DOS</td>
</tr>
<tr>
<td>(-$217E + Top of mem.)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Monitor Output Register: Locations 54-55 ($36-$37)**

<table>
<thead>
<tr>
<th>When Register</th>
<th>To the</th>
<th>Then subsequent output goes to</th>
</tr>
</thead>
<tbody>
<tr>
<td>contents are set by</td>
<td>value</td>
<td></td>
</tr>
<tr>
<td>RESET</td>
<td>-523</td>
<td>Monitor Output Routine</td>
</tr>
<tr>
<td>@ CTRL-P [Note 1]</td>
<td>($F7F)</td>
<td>Monitor Output Routine to TV screen</td>
</tr>
<tr>
<td>PR#s [Note 2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>CTRL-P [Note 1]</td>
<td>49132 + s*256</td>
</tr>
<tr>
<td>PR#s [Note 2]</td>
<td>($C966)</td>
<td>If slot #s contains disk controller, then boot DOS</td>
</tr>
<tr>
<td>[ where s&gt;0 ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOS boot</td>
<td>-$514 + Top of mem.</td>
<td>DOS</td>
</tr>
<tr>
<td>(-$2142 + Top of mem.)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note 1:** The commands s CTRL-K and s CTRL-P are Monitor commands. To type CTRL-K (which does not appear on the TV screen), type K while holding down the CTRL key.

**Note 2:** When DOS is in effect, this command will affect the contents of the Apple Monitor register only if the command is issued as an instruction in a stored program and not in a PRINT CTRL-D instruction.

**Note 3:** In addition to the commands mentioned in Table 1, directly FORSING appropriate values into the Apple Monitor register locations can also be used to select input and output devices, or to re-connect a disconnected DOS.
TABLE 2: DOS INPUT AND OUTPUT REGISTERS

DOS Input Register

<table>
<thead>
<tr>
<th>When Register contents are set by</th>
<th>To the value</th>
<th>Then subsequent input comes from</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOS boot</td>
<td>-741</td>
<td>Monitor Input Routine</td>
</tr>
<tr>
<td>RESET 3D0C</td>
<td>($FDF8)</td>
<td>from Apple keyboard</td>
</tr>
<tr>
<td>INF# [Note 4]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRINT DS;&quot;INF#&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Note 5]</td>
<td></td>
</tr>
<tr>
<td>INF# [Note 4]</td>
<td>49152 + s*256</td>
<td>Slot #s</td>
</tr>
<tr>
<td>PRINT DS;&quot;INF#&quot;</td>
<td>($CO00)</td>
<td>IF slot #s contains a disk controller, then reboot DOS</td>
</tr>
<tr>
<td></td>
<td>[Note 5]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DOS Output Register

<table>
<thead>
<tr>
<th>When Register contents are set by</th>
<th>To the value</th>
<th>Then subsequent output goes to</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOS boot</td>
<td>-528</td>
<td>Monitor Output Routine</td>
</tr>
<tr>
<td>RESET 3D0C</td>
<td>($FDF8)</td>
<td>to the TV screen</td>
</tr>
<tr>
<td>PRINT [Note 4]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRINT DS;&quot;PR#&quot;</td>
<td>[Note 5]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRINT [Note 4]</td>
<td>49152 + s*256</td>
<td>Slot #s</td>
</tr>
<tr>
<td>PRINT [Note 5]</td>
<td>($CO00)</td>
<td>IF slot #s contains a disk controller, then reboot DOS</td>
</tr>
</tbody>
</table>

Note 4. When DOS is in effect, this command will not affect the contents of the DOS Input and Output registers if the command is issued as an instruction in a stored program and not in a PRINT CTRL-D instruction. If a program line such as 120 PRINT is executed, the contents of the Apple Monitor Output register will be changed, leaving DOS partially disconnected until the next input.

Note 5. In this command, it is assumed that the string-variable named DS has been assigned the character control-D, or CTRL-D. This character, which does not appear on the screen, is produced by typing D while holding down the CTRL key.

Note 6. 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.

Note 7. In addition to the commands in Table II, directly POKING the appropriate values into the DOS Input and Output register locations can also be used to select input and output devices. However, the specific memory locations of the DOS Input and Output registers change with different system memory sizes and with different versions of DOS. For this reason, a special procedure exists for changing the contents of the DOS Input and Output register locations. It is a two-step procedure:

a) Change the Apple Monitor Input and Output register locations to the values you wish the DOS Input and Output registers to contain. (This may be done by directly POKING the Apple Monitor register locations or by executing INF and PR# non-DOS instructions in a stored program.)

b) CALL 1002  (from the Monitor, you would type $3EAC).

After this CALL, DOS will be re-connected via the Apple Monitor registers, and the previous contents of the Apple Monitor Input and Output registers will appear in the DOS Input and Output register locations. This CALL can also be used to re-connect BOS any time your program needs to disconnect DOS for awhile. See the program on page 151 for an example using this technique.

Note 8. The Monitor commands s CTRL-K or s CTRL-F, when typed on the keyboard, are not recognized by DOS: they affect the Apple Monitor Input or Output registers directly.
INTEGER BASIC CHAIN

Certain applications are most easily implemented by using a series of two or more programs which are LOADED and RUN sequentially. In such circumstances, the second program often needs to use the values of variables and arrays developed by the first program. The usual RUN command erases the first program's variables and arrays when it loads the second program. In Integer BASIC (but not Applesoft) the DOS command CHAIN allows you to load and run a second program without erasing the first program's variables and arrays.

Suppose you've been using an Integer BASIC program called PART ONE. The command

CHAIN PART TWO

will load and run the Integer BASIC program called PART TWO without clearing the values of any variables used in the program PART ONE. The CHAIN command may be issued in immediate-execution mode as shown above, or from within the last lines of the PART ONE program as a DOS command:

20010 DSW"REM CTRL-D
20020 PRINT D$; "CHAIN PART TWO"

The syntax for the command is familiar:

CHAIN $[Se] $[De] $[Ve]

APPLESOFT CHAIN

The CHAIN command works only with Integer BASIC, but if you do not need to pass variables, it is easy to link Applesoft programs to load and run in sequence. In the first program, just include a last line such as

20000* PRINT CHR$(4); "RUN PART TWO"

When this line is executed, it will start up the second program (where the second program is named PART TWO). In the process, the first program and all its variables are erased.

A different procedure must be used in order to load and run a series of Applesoft programs without erasing earlier values of variables and arrays. To chain in Applesoft, you will need to use the machine-language program called CHAIN that is on the DOS System Master diskette.

To chain from a program called PART ONE to a program called PART TWO, you must have the CHAIN program on the same diskette with the program PART TWO (see next page for instructions). Then, simply insert these two lines as the last two lines to be executed in the PART ONE program:

20000 PRINT CHR$(4); "ELOAD CHAIN, A529"
20010 CALL 529"PART TWO"

The two lines may use any line numbers, but they should come one after the other in the program, as indicated. The first line loads the Applesoft chaining ability into the computer. The second line actually does the chaining (but see next page, for warning).
Unless otherwise indicated, DOS commands may be used either in immediate-execution mode or in deferred-execution mode (within a program). However, some text file commands (e.g. READ and WRITE) must be used in deferred-execution mode.

Most DOS commands refer to a named file. A file may be a text (data) file, or a program in Integer BASIC, APPLESOFT or Machine Language. The tables below indicate which file types may be used by each command. The first table lists the commands alphabetically; the second table groups them by associated file type. The commands CATALOG, FP, INT, MAXFILES, MON, NOMON, FR# and IN# are not included because they do not explicitly refer to named files.

### FILE TYPE USE, LISTED BY DOS COMMAND

<table>
<thead>
<tr>
<th>DOS Command</th>
<th>Integer Basic Files Only</th>
<th>Applesoft Basic Files Only</th>
<th>Sequential Text Access Files</th>
<th>Random Access Files</th>
<th>Machine Language Files Only</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPEND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLOAD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRUN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSAVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHAIN</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXEC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INIT</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOCK</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>OPEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POSITION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RENAME</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>RUN</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAVE</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNLOCK</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>VERIFY</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: use these commands only in deferred execution mode:
- APPEND, OPEN, POSITION, READ, WRITE

### FILE TYPE USE, LISTED BY FILE TYPE

- **Integer Basic files only**
  - CHAIN

- **Integer Basic or Applesoft files**
  - INIT
  - LOAD
  - SAVE
  - RUN

- **Sequential Text files only**
  - APPEND
  - EXEC
  - POSITION

- **Either Sequential Text files or Random-Access Text files**
  - OPEN
  - CLOSE
  - READ
  - WRITE

- **Machine Language files only**
  - BREAD
  - BRUN
  - BSAVE

- **All Types of Files**
  - DELETE
  - LOCK
  - UNLOCK
  - RENAME
  - VERIFY

Note: these commands must be used in deferred-execution mode:
- APPEND, OPEN, POSITION, READ, WRITE
When DOS detects an error connected with disk usage, it normally displays a message describing the error and stops any program that is running. These messages are in addition to the usual messages generated by Applesoft or Integer BASIC. DOS messages can be distinguished from those of Applesoft or Integer BASIC as follows:

An Applesoft message, such as **SYNTAX ERROR**
is preceded by a question mark.

An Integer BASIC message, such as **SYNTAX ERR**
is preceded by three asterisks.

A DOS message, such as

**SYNTAX ERROR**
is preceded by no character at all.

A DOS message appears exactly the same, whether you are in Applesoft, Integer BASIC or the Monitor at the time the message is generated.

If a DOS message occurs when you are using the Monitor, the system is reset to the type of BASIC from which you entered the Monitor.

By using Applesoft's ONERR GOTO command (see the Applesoft manual), you can create Applesoft error-handling routines that deal with DOS messages which would normally interrupt your program. When a DOS error occurs following an ONERR GOTO command in an Applesoft program, a code number for the type of error is stored in decimal memory location 222. This is the same memory location in which Applesoft stores the code for an Applesoft error message. The command

Y = PEEK(222)

sets the value of Y to the Applesoft ONERR GOTO code corresponding to the error that caused an Applesoft ONERR GOTO jump to occur.

DOS messages and their corresponding Applesoft ONERR GOTO codes are shown below, with the most common cause of each message. Each of the messages is then discussed in greater detail, with a more comprehensive list of causes and cures.

### ONERR GOTO Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>DOS Message</th>
<th>Most Common Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LANGUAGE NOT AVAILABLE</td>
<td>Applesoft not on diskette</td>
</tr>
<tr>
<td>2, 3</td>
<td>RANGE ERROR</td>
<td>Command parameter too large</td>
</tr>
<tr>
<td>4</td>
<td>WRITE PROTECTED</td>
<td>Write-protect tab on diskette</td>
</tr>
<tr>
<td>5</td>
<td>END OF DATA</td>
<td>Reading beyond end of text file</td>
</tr>
<tr>
<td>6</td>
<td>FILE NOT FOUND</td>
<td>File misspelled, or not on diskette</td>
</tr>
<tr>
<td>7</td>
<td>VOLUME MISMATCH</td>
<td>Wrong Volume parameter</td>
</tr>
<tr>
<td>8</td>
<td>I/O ERROR</td>
<td>Door open, or diskette not INITed</td>
</tr>
<tr>
<td>9</td>
<td>DISK FULL</td>
<td>Too many files on diskette</td>
</tr>
<tr>
<td>10</td>
<td>FILE LOCKED</td>
<td>Attempt to over-write a LOCKed file</td>
</tr>
<tr>
<td>11</td>
<td>SYNTAX ERROR</td>
<td>Bad file name, parameter, or comma</td>
</tr>
<tr>
<td>12</td>
<td>NO BUFFERS AVAILABLE</td>
<td>Too many text files OPEN</td>
</tr>
<tr>
<td>13</td>
<td>FILE TYPE MISMATCH</td>
<td>Diskette file doesn't match command</td>
</tr>
<tr>
<td>14</td>
<td>PROGRAM TOO LARGE</td>
<td>Insufficient Apple memory available</td>
</tr>
<tr>
<td>15</td>
<td>NOT DIRECT COMMAND</td>
<td>Command must be in a program</td>
</tr>
</tbody>
</table>

### Discussion

**LANGUAGE NOT AVAILABLE (ONERR GOTO code = 1)**

Occurs if DOS cannot find a programming language, either Integer BASIC or Applesoft, that is required to execute a DOS command. The commands FF, INT, LOAD and RUN may all initiate a language search. If Integer BASIC is requested, DOS looks for that language in RAM. If Applesoft is requested, DOS first looks for the language in ROM, using Applesoft from an Applesoft firmware ROM card (if available) regardless of the card's switch position. If Applesoft is not found in ROM, DOS looks on the diskette in the 'default' disk drive -- the drive indicated by the default or most recent values of the S and D parameters. DOS will not look on any other disk drive.

This message usually arises after a DOS request for diskette Applesoft, if the diskette in the default drive does not contain the program APPLESOFT. Replace the diskette with one that contains the program APPLESOFT; or use the D parameter with any DOS command, to select the other drive. A command such as this will do nicely:

**FF, D2**

If you think DOS should have found Integer BASIC in ROM, but it didn't, try the following:

1. **EACH OFF your Apple and remove the cover.**
2. Locate the row of four large ROM chips (black, rectangular objects) in the middle of the main printed-circuit board. These chips are labeled "ROM F6", "ROM F5", "ROM E8" and "ROM E7".
3. Press down firmly on these chips.
4. Replace the cover, turn on the Apple and try INT again.

If you think DOS should have found Applesoft on your firmware ROM card, but it didn't, try the following:

1. **TURN OFF your Apple and remove the cover.**
2. Unplug the Applesoft firmware ROM card. Locate the row of five large ROM chips (black, rectangular objects) across the card. These chips are labeled 1, 2, 3, 4, and 5 above the chips, and D6, D5, D4, E6, and E5 below the chips.
3. Press these chips firmly into their sockets.
4. **PLUG the Applesoft card back into slot F6, the leftmost slot.**
5. Replace the cover, turn on the Apple and try FF again.
RANGE ERROR (CERRR GOTO code = 2 or 3)

Occurs when the value of a DOS command parameter or a DOS command quantity is too large or too small. Refer to the manual to see which DOS commands are used with which parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Letter</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Files:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slot</td>
<td>S</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>Drive</td>
<td>D</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Volume</td>
<td>V</td>
<td>0*</td>
<td>254</td>
</tr>
<tr>
<td>Sequential Text Files:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte</td>
<td>B</td>
<td></td>
<td>32767</td>
</tr>
<tr>
<td>Relative Field</td>
<td>R</td>
<td></td>
<td>32767</td>
</tr>
<tr>
<td>Absolute Field</td>
<td>R</td>
<td></td>
<td>32767</td>
</tr>
<tr>
<td>Random-Access Text Files:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record Length</td>
<td>L</td>
<td>1</td>
<td>32767</td>
</tr>
<tr>
<td>Record Number</td>
<td>R</td>
<td>0</td>
<td>32767</td>
</tr>
<tr>
<td>Binary Files:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Starting Address</td>
<td>A</td>
<td>0</td>
<td>65535</td>
</tr>
<tr>
<td>Number of Bytes</td>
<td>L</td>
<td>1</td>
<td>32767</td>
</tr>
<tr>
<td>DOS Command</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PH# s</td>
<td>s</td>
<td>s</td>
<td>16 **</td>
</tr>
<tr>
<td>IN# s</td>
<td>s</td>
<td>s</td>
<td>16 **</td>
</tr>
<tr>
<td>MAXFILES n</td>
<td>n</td>
<td>1</td>
<td>15</td>
</tr>
</tbody>
</table>

* Minimum volume number INIT will actually assign to a diskette is 1.
** Maximum slot number built into the Apple II is 7. In deferred-execution mode only, the SYNTAX ERROR message is given for s values from 8 through 10.

Note: The use of values outside the above ranges does not always cause the RANGE ERROR message. Any DOS command parameter or command quantity that is less than 0 or greater than 65535 will cause the SYNTAX ERROR message, not the RANGE ERROR message.

WRITE PROTECTED (CERRR GOTO code = 4)

Occurs when DOS attempts to store information on a diskette, but the disk drive does not detect a "write-protect" notch or cutout on the left side of the diskette’s outer case. The following are the most likely causes:

1. There is an adhesive label placed over the diskette’s write-protect cutout, to prevent accidentally overwriting or deleting any information on the diskette. This label may be removed, whereupon DOS will SAVE or BSSAVE or WRITE to the diskette.

2. There is no write-protect cutout on the diskette. This is true on the System Master diskette, for maximum protection. While not recommended, it is possible to carefully cut a notch of exactly the correct size and in exactly the correct place. Use another diskette’s write-protect notch for a model.

3. If you receive this message while RUNNING the COPY program, and the cause is not either 1 or 2, above, you may have inserted the diskette into the drive incorrectly (in any other situation, DOS gives the I/O ERROR message to signal incorrect diskette insertion). Check the diskette’s position in the drive, and re-read Chapter 1’s discussion on inserting diskettes.

END OF DATA (CERRR GOTO code = 5)

Occurs when you try to retrieve information from a portion of a text file where no information has ever been stored. Any byte beyond the last field in a sequential text file, or beyond the last field of each record in a random-access text file, may contain the value $0. Zero is the ASCII code for a null character, a “nothing”, and any command that causes the retrieval of this character results in the END OF DATA message. Remember that only OPEN automatically sets the position-in-the-file pointer back to the file’s beginning. The message usually occurs after an INPUT or a GET command, and can arise in several different ways:

1. Too many successive INPUTS or INPUTs with too many variables. Each INPUT or INPUT variable causes one additional, adjacent field to be read into the Apple.

2. Too many successive GETs. Each GET reads one additional, adjacent byte or character into the Apple.

3. The B (for Byte) parameter was too large. In sequential files, this parameter must not specify a byte beyond the last RETURN character in the file. In random-access files, the B parameter should not specify a byte beyond the last RETURN character in the currently selected record. Remember, the first byte in a file is a 0, or a record is byte 0.

4. The R (for Relative-field position) parameter in a POSITION command was too large. In sequential files, this parameter must not specify a field beyond the last existing field in the file. In random-access files, POSITION’s R parameter should not specify a field beyond the last existing field in the currently selected record. Remember, the R parameter used with POSITION is not the same as the R parameter used with READ. It specifies a field position in the file, relative to the current file position and forward in the file, only. If R specifies no change in the current file position, RJI jumps the file position ahead to the first byte following the field that contains the current position.

POSITION scans forward through the contents of the file, byte by byte, looking for the Bp-th RETURN character. If it encounters a 0 byte (the null character) before finding the required RETURN character, the END OF DATA message is given immediately. It is not necessary actually to INPUT or GET the null character.
5. The R (for absolute-field position) parameter in an EXEC command was too large. This parameter may specify the first field beyond the last existing field in a file, but attempting to specify the second field beyond the file's end will cause the END OF DATA message. Remember, R$ specifies the first field in a file.

6. The R (for Record) parameter in a READ command specifies a random-access file record in which nothing has yet been stored. Before you can READ from a particular record in a random-access file, you must first WRITE some information into that record.

Remember, READ's R parameter is not the same as the R parameter used by POSITION or EXEC. READ's R parameter specifies an absolute record in a file; R$ is the file's first record, and so on.

DOS uses the OPEN command's L parameter for calculating where the R-th record begins, so the OPEN preceding READ must use the same L parameter value as the OPEN that preceded WRITE for that file.

FILE NOT FOUND (ONERR GOTO code = 6)

Occurs when certain DOS commands specify a file name that is not in the CATALOG for the diskette in the selected or default disk drive. Only the commands SAVE, BSAVE, INIT and OPEN can create a file whose name did not previously exist. In addition to these, CLOSE may be used with any valid name. A file name specified by any other DOS command must already exist on the diskette.

This message may arise in various ways: 1. You may have misspelled the file's name, by a typing error or by omitting the comma that separates the file name from a following parameter. Check the CATALOG for the exact spelling of the file's name. Warning: If you have accidentally typed control characters into the name of a file, CATALOG will not display those characters. For help, see "File Names" in Appendix F.

2. The file is on another diskette. Check the CATALOG.

3. The file has been accidentally DELETED. Check the CATALOG.

4. When you use the INIT command on a diskette, you specify a file name which DOS thereafter attempts to put on each time you boot the system with that diskette in disk drive 1. Unless you write a BASIC program, and save it using the name given to INIT, the FILE NOT FOUND message will be given each time the system is booted with that diskette in drive 1. If you can't remember the name of this "greeting program", you must re-INIT the disk, or run MASTER CREATE to rename the greeting program without destroying the contents of the diskette.

VOLUME MISMATCH (ONERR GOTO code = 7)

Occurs when the Volume (V) parameter used in a DOS command is not the same as the volume number assigned to the diskette in the default or selected disk drive, when that diskette was INITIALIZED. The volume number of a diskette is shown at the head of each CATALOG display. Unless a DOS command specifies a particular volume, the diskette's volume number is ignored, and no message is given. If a DOS command specifies volume $, the diskette's volume number is still ignored. If no volume number is given with INIT, or if volume number $ is given, the diskette will be initialized with the default volume number 254.

1/0 ERROR (ONERR GOTO code = 8)

Occurs after an unsuccessful attempt to store data on a diskette or to retrieve data from a diskette (DOS tries 96 times, then gives up). This message can occur in the following ways:

1. The selected or default drive's door is open. Close the door to the disk drive.

2. No diskette in the selected or default disk drive. Put a diskette into the drive and close the drive door.

3. Diskette in the selected or default disk drive has not been INITIALIZED. INIT the diskette.

4. Diskette is inserted incorrectly. Check the diskette, and re-read the section in Chapter 1 on inserting diskettes.

5. During execution of a VERIFY command, DOS found the specified file was not stored correctly on the diskette. If the file's information is still in memory, try storing it again (perhaps on a different diskette).

6. The DOS command's D (Drive) parameter has specified a disk drive that does not exist in this system. The default drive is the non-existent drive. Just specify the correct D parameter with the next DOS command to reset the default.

7. The system is trying to access a 13-sector diskette using 16-sector DOS. Use the MUFFIN program to update your diskette to 16-sectors.

8. The DOS command's S (Slot) parameter has specified a slot that does not contain a disk controller card in this system.

You are in trouble, but can still save yourself; don't despair. The problem is that the default slot is now the empty slot your last DOS command specified. The next DOS command without a slot parameter will go to the empty slot and return the same message as before. To reset the slot correctly,

a) Press RESET
b) If you see a Monitor prompt (*), type 300;
c) Type CATALOG S0 where 0 is the correct slot.
DISK FULL (ERRNO GOTO code = 9)

Occurs when DOS attempts to store information on a diskette, and finds that no more storage space is available on that diskette. A maximum of 256 sectors can be filled with user-stored information, so displayed in the CATALOG (if an individual file exceeds 255 sectors, the CATALOG display of its length starts over again at 000). If you receive the DISK FULL message, rest assured that all files are CLOSED, and that DOS saved for you all it could (leaving you with some portion of your file not on the diskette). If you receive this message while saving a file called STUFF, the first thing you should do is to DELETE STUFF and then save your program on another diskette that has more room left.

FILE LOCKED (ERRNO GOTO code = 10)

Occurs when you try to SAVE, BSAVE or WRITE using a file name that has been LOCKed on the diskette that is in the selected or default drive. Check the CATALOG display: the names of LOCKed files are preceded by an asterisk (*) in the CATALOG display. A file is LOCKed to prevent accidental over-writing. Use another diskette or UNLOCK the desired file.

SYNTAX ERROR (ERRNO GOTO code = 11)

Occurs when DOS encounters a syntax error in a DOS command. Check the manual for the exact syntax required for the command is question. The problem may be a non-valid file name (see Appendix F), an incorrect parameter symbol, a missing parameter, a missing or incorrect separator (usually a comma). This message will also arise if a parameter value or command quantity is in a negative number or is greater than 65535.

Rarely, every DOS command causes the Applesoft or Integer BASIC Syntax Error message. This usually means that DOS has not been booted or has become "disconnected" from input and output. Try pressing the RESET key, then typing 3D0C to reconnect DOS; or, re-boot the disk.

NO BUFFERS AVAILABLE (ERRNO GOTO code = 12)

Occurs when a DOS command requires another file buffer for input or output, and all the available file buffers are already in use. On booting the system, DOS reserves enough space in the Apple's memory for three input-and-output file buffers. A subsequent MAXFILES command can increase or decrease the number of available file buffers, and a CLOSE command can release file buffers currently in use for text files.

Many DOS commands use one file buffer for input or output during their execution, and then relinquish that buffer when execution of the command has ceased.

When a text file in OPENed, a file buffer is assigned to that file for input and output. This buffer remains in use, generally, until its file is CLOSED either specifically by file name or by the nameless CLOSE that de-alloclates all the text-file buffers. A text file is not automatically CLOSED by a program's coming to an end. To conserve buffer space, CLOSE files as soon as you are through using them. Remember that the next OPEN will re-set the position-in-the-file pointer to the file's beginning.

The MAXFILES command can be used to increase buffer space before writing the program or loading the program into memory. Increasing MAXFILES moves HIMEN down, and this can erase stored Integer BASIC program lines or Applesoft strings. Changing MAXFILES in the middle of a program can be especially dangerous.

FILE TYPE MISMATCH (ERRNO GOTO code = 13)

Occurs when a DOS command attempts to use a file name that is already assigned to a file whose file type is inappropriate to the present command. If you are sure the command is correct, use a file name that is not now on the diskette, use a different diskette, RENAME the existing file or DELETE the existing file.

This message arises from several different incorrect combinations of DOS commands with existing file types. Here are the correct combinations:

LOAD f, RUN f, SAVE f
CHAIN f
OPEN f, READ f, WRITE f,
APPEND f, POSITION f, EXEC f
BLOAD f, BRUN f, BSAVE f

The greeting program's file name, specified with INIT or UPDATE, must refer to an Applesoft or Integer BASIC program file.
APPENDIX C

FORMAT OF DISKETTE INFORMATION

124 Overview of the Storage Process
126 WRITING into a Sequential Text File
126 WRITING into a Random-Access Text File
126 How DOS WRITES into Text Files: General Procedure
127 Contents of File Sections
129 The Track/Sector List
129 The Diskette Directory
132 Volume Table of Contents
132 Track Bit Map
135 Track and Sector Allocation Order
136 Retrieving Information from the Disk
136 Reading from a Sequential File
137 READING from a Random-Access File

PROGRAM TOO LARGE (OBERG GOTO code = 14)

Occurs when a DOS command attempts to place a diskette file into Apple’s memory, and finds the available memory insufficient to contain the entire file. You (or a previous program) may have set HIMEM too low for the current task, or a large MAXFILES may have set HIMEM too low. If you set the number of file buffers to three, using the command MAXFILES 3 then HIMEM will be returned to the booted value given in Appendix D, Table 2.

If you are in Integer BASIC, and HIMEM is set low (to protect the high-resolution screen memory, for instance), you may experience trouble on shifting to diskette Applesoft. Diskette Applesoft occupies about 12.5K of memory, but a shift to diskette Applesoft (with PP or LOAD or RUN) does not reset HIMEM to maximum. When DOS tries to load the Applesoft program from diskette, the message PROGRAM TOO LARGE will be given if HIMEM is below about 13104. The system will be left in integer BASIC again, and you must set HIMEM higher from integer BASIC. See Appendix D, Table 2 for your system’s maximum HIMEM with DOS and three file buffers.

In deciding whether or not a program will fit into the available memory, DOS looks only at the number of diskette sectors occupied by the program. In general, the program does not completely fill the last sector (256 bytes), but DOS ignores this fact. DOS compares only the high-order byte of LOMEM (Integer BASIC) or HIMEM (Applesoft) with the high-order byte of the projected end-of-program location. Thus a program which should fit into memory, but which would leave less than 256 bytes of free memory after loading, may cause the PROGRAM TOO LARGE message. Sometimes this can be corrected by moving HIMEM or LOMEM slightly, to change the high-order byte, before loading the program.

NOT DIRECT COMMAND (OBERG GOTO code = 15)

Occurs when you try to use one of the text file commands APPEND, OPEN, POSITION, READ or WRITE from immediate-execution mode. These DOS commands can be used only from within PRINT statements in program lines.
This appendix tells how information is stored on a diskette, and how DOS remembers where particular information has been stored.

In the following discussion, a dollar sign ($) or the label "Hex" preceding a number indicate that the number is expressed in hexadecimal.

OVERVIEW OF THE STORAGE PROCESS

In the Disk II system, information is recorded on a diskette in 35 concentric zones or bands, called tracks. These tracks are numbered from track #0, the outermost, through track #32, the innermost. The disk drive's recording and reading head can be moved in and out, to stop and hover over each of these 35 different zones of the spinning diskette.

Furthermore, the length of each track on the diskette is divided into 16 segments, called sectors. These sectors are numbered from 0 through 15, and up to 256 ($100) bytes of information can be stored in each sector. Once the disk drive's recording and reading head is positioned over a given track, that track's 16 sectors will pass under the head, one after the other, each time the diskette spins around. DOS always records information on the diskette in 256-byte chunks, exactly filling one sector each time.

To store information on the diskette, DOS first puts 256 bytes (one sector's worth) of the information in an area of Apple's memory called a file buffer. When this file buffer is full, the information is stored in one sector on the diskette. Then DOS fills Apple's file buffer with the next 256 bytes of information and stores that information on the diskette.

In general, DOS will begin storing a program or text file whenever it can find an unused sector on the diskette. When that sector is filled with its 256 bytes of information, 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 remember which sectors of which tracks contain the information for a particular file, DOS makes up a list of each track and sector used, as it stores the file. Then DOS stores that list, called a track/sector list, in yet another free sector (or sectors) on the diskette.

Finally, the file's name, file-type, length in sectors, and the diskette's location of the file's track/sector list are recorded in a special area of track #1 called the directory. At this time, too, the diskette's track bit map is updated to correctly show which sectors of each track are currently in use.

WRITING INTO A SEQUENTIAL TEXT FILE

Entries in a text file consist of 1 to 32767 characters stored as their equivalent ASCII codes and ended by a RETURN character (either ASCII $0D or ASCII $0D). Each such entry is called a field.

In a sequential text file (no Length parameter specified when the file was OPENed), fields are stored immediately following each other (see Chapter 6). DOS writes the first byte of each new field immediately following the RETURN character that ended the previous field (unless otherwise instructed by a Byte parameter). Each time the file is OPENed, DOS forgets the current position within the file, and starts WRITing again in byte 0 (again, unless otherwise instructed by a Byte parameter).

In order to rewrite a particular field or character within a sequential file, WRITE can be used with the R (for Byte) parameter to begin writing at the specified absolute byte of the file (the first byte in the file is byte 0, the next is byte 1, etc.). The byte specified may be before or after the current position in the file.

It is very difficult to remember exactly which character appears in every byte of a text file, especially in a sequential text file. For this reason, use of the Byte parameter in sequential text files is not recommended.

The POSITION command can be used with an R (for Relative-field) parameter to move a pointer ahead (only) through the file a specified number of fields relative to the current position in the file. A program portion such as

```
129 PRINT D$: "OPEN NAMES"
130 PRINT D$: "POSITION NAMES, R1 2"
140 PRINT D$: "WRITE NAMES"
150 PRINT D$: "APPLE COMPUTER"
159 PRINT D$: "CLOSE NAMES"
```

will attempt to WRITE the characters APPLE COMPUTER into the NAMES file, beginning in the first byte of the fourteenth field (the first field in Relative-field $0).

POSITION can move you to the first byte of any given field Relative to the current position in a sequential text file. If you then re-WRITE that field, however, you must make sure that you re-PRINT exactly the same number of characters that you PRINTed in that field originally. If you PRINT fewer characters, you will have created two new fields: the field you just PRINTed, and the tail-end of the original field you were over-writing. If you PRINT more characters than the original field contained, you will have over-written some of the characters at the start of the next field.
WRITING INTO A RANDOM-ACCESS TEXT FILE

For a random-access text file, a Length parameter is specified when the file is OPENed. The Length parameter determines the number of bytes in a record, which is a field or a collection of fields that DOS treats as a unit. Each record in a random-access text file is like a separate sequential text file whose maximum total length has been specified by the Length parameter. As long as you stay within that maximum length, you can WRITE and re-WRITE all you want, without affecting any other record in the file. WRITE can be used with the R (for Record) and B (for Byte) parameters to begin writing into any byte of a specified record.

Since any DOS command will terminate WRITE-ting, you cannot use POSITION to jump ahead into different fields within the record specified by the WRITE command.

DOS uses the Length parameter to calculate where to write the first byte of each new record (i.e., bytes beyond the first byte of the previous record). DOS simply skips over any bytes between the previous record’s last character and byte 0. The bytes skipped over will continue to contain whatever values were stored there at some earlier time (see the next section for details).

If you attempt to WRITE more characters in a random-access record than you specified in the Length parameter, all the characters are stored correctly on the diskette. However, when you begin WRITE-ting to the next record, DOS calculates the new record’s starting position as if the previous record had been within the specified Length. The new record then overwrites the last characters of the previous, over-sized record, including the end-marking RETURN character of the previous record’s last field. The result is messy.

HOW DOS WRITES INTO TEXT FILES: GENERAL PROCEDURE

When you WRITE a field into a text file, DOS first checks on the diskette to see whether or not you have already stored information in the sector which should contain that field. If your file has never used that sector before, DOS places zeros in all 256 bytes of an Apple file buffer, and then lets you put your information into that buffer for later storage in the correct diskette sector. The contents of the file buffer are stored on the diskette when your information has completely filled 256 bytes of the buffer, or when the file is CLOSED.

Thus, when you WRITE to a particular sector the first time, unused bytes are given the value zero. An attempt to READ a byte containing a zero (the ASCII code for the null character) will result in the message END OF DATA.

But if DOS finds your file has already stored information in the sector which should contain the field that you are now WRITE-ting, it reads all 256 bytes from that sector into the Apple’s file buffer. After you have changed any of these file-buffer bytes to contain your new information -- the WRITE, POSITION (sequential files only) and PRINT commands take care of this for you -- DOS then stores the buffer’s contents right back into the the same diskette sector where they originated. The contents of the file buffer are stored back on the diskette when you attempt to change any byte not in the sector that was read into the file buffer, or when the file is CLOSED.

Thus, if you WRITE more information for a file, and DOS stores that information in a diskette sector already being used by your file, this will not re-write any zeros in unused bytes. Any of those sector bytes which were not re-write will continue to contain whatever information might have been stored there before your WRITE command. This is true of the unused bytes at the end of a sequential text file, and also true of the unused bytes in each fixed-length record of a random-access text file.

CONTENTS OF FILE SECTORS

Now that you know the general process of recording a file on diskette, we can discuss each element in more detail. The actual information stored, sector by sector, is different for each type of file.

<table>
<thead>
<tr>
<th>Format of File Sectors for different file types</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File type</strong></td>
</tr>
<tr>
<td>BASIC (both types)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Text</td>
</tr>
</tbody>
</table>

126
### The Track/Sector List

As a file is stored on the diskette, DOS makes a list of the diskette locations used by the file. This track/sector list is then stored on the diskette in the same way the file itself was stored. The contents of a track/sector list are as follows:

<table>
<thead>
<tr>
<th>File type</th>
<th>Sector</th>
<th>Contents of byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>1st sector</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Starting RAM address, low byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot; &quot; &quot; &quot; high byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Length of RAM image, low byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot; &quot; &quot; &quot; high byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Binary data</td>
</tr>
</tbody>
</table>

**THE TRACK/SECTOR LIST**

If any track/sector pair is $0/0$ this indicates an unassigned sector (usually the end of the file, although text files may contain $0/0$ indicators for many as-yet-unassigned sectors where future bytes or records may be written).

Subsequent sectors of the track/sector list (if the list extends beyond 122 track/sector pairs) are identical to the first sector described above, except that the track/sector pairs refer to subsequent groups of 122 file sectors. Also, Link bytes 1 and 2 will be different for each subsequent sector. Each Link pair gives DOS the diskette location of the next portion of the track/sector list. If both bytes of the Link are $0$, this indicates the final portion of the track/sector list.

With a text file, 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, filling unassigned track/sector pairs with zeros.

Thus, if the Length parameter for a random-access file is 128 (two records per sector) and you WRITE only to record number 2700, thirteen diskette sectors are used: one for the contents of record number 2700, and twelve for the sectors of the track/sector list. The contents of records number $0$ through 2698 may someday occupy 1342 sectors; but until those records are written, they do not use any diskette space. The track/sector list giving the locations of the sectors containing records number $0$ to 2693 occupy eleven sectors.

### The Diskette Directory

On every INITIALIZED diskette, track 011 is reserved for information concerning the contents of the diskette. This is where DOS stores the directory containing, for each file, the file's name, its file type, the number of sectors occupied by the file (MOD 256), and the diskette location of the file's track/sector list. The CATALOG command causes most of this information to be displayed on the screen. Each sector of a diskette directory is formatted as follows:

<table>
<thead>
<tr>
<th>File type</th>
<th>Sector</th>
<th>Contents of byte</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Link: track number where continuation of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>track/sector list may be found.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Link: sector number where continuation of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>track/sector list may be found.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(If both bytes of Link = $0$, no link.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sector base number (counts groups of 122 sectors)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Track number of first file sector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sector &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Track number of second file sector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sector &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Track number of third file sector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sector &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Track number of 122nd file sector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sector &quot; &quot; &quot; &quot;</td>
</tr>
</tbody>
</table>

128 129
### Byte (Hex) | Contents of byte
---|---
3 through A | Not used
8 through 2D | Directory entry for file 1 (see below)
2E through 5F | Directory entry for file 2
51 through 73 | Directory entry for file 3
74 through 95 | 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

The file numbers shown for the seven directory entries are arbitrary. When a file is DELETED, DOS marks the directory entry for that file (see following table). The next time a file is stored, DOS replaces the old marked directory entry with the directory entry for the new file. Thus, while DOS originally fills the directory in the order shown, file DELETions soon render this order meaningless.

The diskette directory begins in track $11, sector $FF. If more space is needed to store additional directory entries, sector $FF is linked to sector $FE. If still more space is needed, sector $FE is linked to sector $D, and so on, through sector $11. This allows the directory to store directory entries for a maximum of 105 different files.

Each directory entry is written in the following format:

#### DIRECTORY ENTRY FOR ONE FILE

<table>
<thead>
<tr>
<th>Relative Byte (Hex)</th>
<th>Contents of byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Track number of the file's track/sector list (The original value is copied into $20 and the value of byte 0 is changed to $FF when the file is DELETED.)</td>
</tr>
<tr>
<td>1</td>
<td>Sector number of the file’s track/sector list</td>
</tr>
<tr>
<td>2</td>
<td>File type (see discussion on the next page)</td>
</tr>
<tr>
<td>3 through 20</td>
<td>File name</td>
</tr>
<tr>
<td>21 through 22</td>
<td>Sector count: the number of diskette sectors (MOD 256) occupied by the file</td>
</tr>
</tbody>
</table>

A directory entry's relative byte 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 listed in the previous table).

Because only one byte is used to store a file’s sector count, the maximum directory sector count is 255 (87F). If a file exceeds 255 sectors, its sector count (as displayed by CATALOG) starts over again at $000. This does not affect use of the file, but may give an erroneous impression of how full the diskette is.

The eight bits of a file's type-designating byte, relative byte number 2 in a file's directory entry (see previous table), are assigned values as follows:

### BYTE INDICATING THE FILE TYPE

<table>
<thead>
<tr>
<th>Bit</th>
<th>Symbol</th>
<th>File type designated</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>*</td>
<td>File is locked (write protected) if this bit = 1 File is unlocked (not protected) if this bit = 0</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Expansion type for future use (normally zero)</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>Binary file if this bit = 1</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>Applesoft BASIC file if this bit = 1</td>
</tr>
<tr>
<td>0</td>
<td>T</td>
<td>Integer BASIC file if this bit = 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Text file if bits 0 through 6 are all zero</td>
</tr>
</tbody>
</table>

The file type is determined by a 1-bit appearing in one of the bits 0 through 4. If bits 0 through 6 are all 0-bits, the file type defaults to a Text file.

The file's type-designating byte can thus take on the following values:

### VALUES FOR BYTE INDICATING FILE TYPE

<table>
<thead>
<tr>
<th>File type</th>
<th>Value of Type byte (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>0</td>
</tr>
<tr>
<td>Integer</td>
<td>1</td>
</tr>
<tr>
<td>Applesoft</td>
<td>2</td>
</tr>
<tr>
<td>Binary</td>
<td>4</td>
</tr>
</tbody>
</table>
VOLUME TABLE OF CONTENTS

Sector 00 of track 011 contains the diskette's Volume Table of Contents, or VTOC. The VTOC stores the following information:

<table>
<thead>
<tr>
<th>Byte (Hex)</th>
<th>Value (Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>Not used</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>Track number of first directory sector</td>
</tr>
<tr>
<td>2</td>
<td>8F</td>
<td>Sector &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>DOS release number</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>Not used</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>6</td>
<td>1 through FF</td>
<td>Diskette volume number (default: 0FE)</td>
</tr>
<tr>
<td>7 through 26</td>
<td>0</td>
<td>Not used</td>
</tr>
<tr>
<td>27</td>
<td>7A</td>
<td>Minimum number of track/sector pairs possible in each sector of a track/sector list</td>
</tr>
<tr>
<td>28 through 2F</td>
<td>0</td>
<td>Not used</td>
</tr>
<tr>
<td>30</td>
<td>FF</td>
<td>These four bytes are a mask for the track bit maps (see next 2 pages): each 1-bit enables one of the 16 sectors to be used in every track.</td>
</tr>
<tr>
<td>31</td>
<td>FF</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>23</td>
<td>Number of tracks per diskette</td>
</tr>
<tr>
<td>35</td>
<td>0F</td>
<td>Number of sectors per track</td>
</tr>
<tr>
<td>36</td>
<td>00</td>
<td>Number of bytes per sector, low byte &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>37</td>
<td>01</td>
<td>high byte &quot; &quot; &quot; &quot;</td>
</tr>
</tbody>
</table>

7C through 7F | 0 | Track $11 bit map (Directory & VTOC) |

80 and 81 | ? | Track $12 bit map |

62 and 63 | ? | |

C0 and C1 | ? | Track $22 bit map |

C2 and C3 | ? | |

C4 through FF | 0 | Not used |

TRACK BIT MAP

Starting in byte $18 of the VTOC (see previous table), subsequent four-byte groups each contain the track bit map for one of the diskette's 35 tracks. The arrangement of 1-bits and 0-bits within a track's bit map shows DOS which sectors of that diskette track are currently in use, and which sectors are free. The bit map for each track uses the following format:

<table>
<thead>
<tr>
<th>Designated Sector</th>
<th>Designated Sector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte (Hex)</td>
<td>Bit (Hex)</td>
</tr>
<tr>
<td>1st</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

3rd & 4th | All | Spare |

[Continued on next page]
If a bit in the track bit map contains the value 1, the sector corresponding to that bit is free. If a bit in the map contains the value 0, the sector corresponding to that bit is currently in use. Bits marked "Spare" in the table above contain the value 0; these bits are not used. The track bit map for a typical track might appear as follows:

**Typical Track Bit Map**

<table>
<thead>
<tr>
<th>1st byte</th>
<th>2nd byte</th>
<th>3rd byte</th>
<th>4th byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 1 1 1</td>
<td>1 1 1 1</td>
<td>1 1 1 1</td>
<td>1 1 1 1</td>
</tr>
<tr>
<td>F E D C B A 9 8</td>
<td>7 6 5 4 3 2 1 0</td>
<td>Spare</td>
<td>Spare</td>
</tr>
</tbody>
</table>

Sectors designated

1 = Free sector (assuming the corresponding bit of the mask, VTOC bytes $30 through $33, is also 1)

0 = Sector in use

When a file is being stored on the diskette (using `WRITE`, `SAVE` or `BSAVE`), an entire track is allocated to the file at once (when possible), and the track's bit map shows the entire track in use. Then, when the file is closed, those sectors not actually used are again designated as free, in the bit map for that track.

Sectors actually used to store a file's information, however, can only be "set free" when that file name is `DELETED`. Suppose your diskette contains a 100-sector BASIC file named BIG, for instance. If you now `SAVE`, on the same diskette, a 2-sector file with the same name BIG (overwriting the old file) a `CATALOG` of the diskette will reveal that your 2-sector file BIG is still using up 100 sectors. To free up unnecessary sectors used by a BASIC file named BIG, use the following sequence of commands:

- `LOAD BIG`
- `DELETE BIG`
- `SAVE BIG`

A similar process can be used to release unnecessary sectors used by binary files.

To release unnecessary sectors being used by a text file, you will have to read each of the file's fields into the Apple. If you store all the fields in an array, you can then `DELETE` the original file before `WRITE`ing each record back onto the diskette using the original file name. Another way to do this is to read each field into the Apple and immediately `WRITE` the field back onto the diskette using a file name that is different from the original file name. When you have read and re-written the last field, you can `DELETE` the original file.

**Track and Sector Allocation Order**

Each diskette contains 35 tracks, three of which are reserved for DOS and one for the Directory, leaving 31 tracks for the user. Each track contains 16 sectors, so all together 496 sectors are available to the user.

Sectors are filled starting with sector $00$ and working back to sector $0F$. Tracks are filled starting with track $S12$ (just inside the directory/VTOC track) and proceeding inward to track $S2$ (the innermost track). When track $S22$ has been filled, tracks are then filled starting with track $S10$ (just outside the directory/VTOC track) and working outward to track $S3$ (the outermost track available to the user).
RETRIEVING INFORMATION FROM THE DISK

To retrieve a file from diskette, DOS follows the process used to store the file, but in reverse. After a command such as
LOAD FILE
or
LOAD FILE
for instance, DOS goes to the diskette’s file directory in track S1, and finds the directory entry containing the name FILE. This entry also contains the diskette location (by track and sector) of the desired file’s track/sector list. DOS then goes to this track/sector list, and reads the first track/sector pair. This pair specifies the diskette location of the first sector containing the program named FILE. When DOS has read that first sector of program into the Apple, it returns to the track/sector list for the location of the program’s second sector, and so on.

READING FROM A RANDOM-ACCESS FILE

The text-reading process is somewhat different when READING from a specified record of a random-access text file (also see WRITING TO A RANDOM-ACCESS FILE in this appendix). In a random-access file, each record is composed of the same number of bytes, specified in the Length parameter when the file was OPENed prior to WRITING the file. When this same file is OPENed prior to READING it, an identical length parameter is given. To find the beginning of a particular record (specified by the READ command’s R parameter), DOS uses the Length parameter to calculate the number of bytes occupied by all the preceding records. That number is then divided by 256 (SIZE) to determine how many file sectors DOS must skip over to reach the sector containing the desired record. Then DOS examines the file’s track/sector list and finds the diskette location of the desired file sector. Finally, DOS reads the correct sector into the Apple’s file buffer. Then the correct bytes can be read from the file buffer.

This same retrieval process would be followed even if the text file had originally been stored as a sequential file, or as a random-access file using a completely different length. DOS blindly calculates the sector and byte position of the requested record according to whatever length parameter you specify when you OPEN the file prior to READING from it, regardless of the length parameter (if any) that was used when writing the file in the first place.

By using the READ command with both R (for Record) and B (for Byte) parameters, you can cause the next INPUT to begin reading from the specified absolute byte in the specified record (each record’s first byte is 0, the next is 1, etc.). This byte may be before or after the current position within the record. To use this parameter effectively, however, you must know the contents of every byte in your file. The POSITION command uses the R (for Relative-field) parameter to move DOS’s current-position pointer the specified number of fields forward (only) through the file, relative to the current position in the file. Each time you OPEN a file, DOS forgets its current position in the file and starts READING again from the beginning of the file (unless otherwise instructed by a byte parameter).

The INPUT command treats a response somewhat differently in Integer BASIC and in Applesoft. If certain characters such as the colon or comma appear in the response field, further characters in the field may be ignored or assigned to multiple INPUT variables (if any). For details, see the appropriate manual for Integer BASIC or for Applesoft.

The INPUT command treats a response somewhat differently in Integer BASIC and in Applesoft. If certain characters such as the colon or comma appear in the response field, further characters in the field may be ignored or assigned to multiple INPUT variables (if any). For details, see the appropriate manual for Integer BASIC or for Applesoft.
### Table 1: Apple II Memory Maps

#### A. Memory Areas Over-written When Booting DOS

<table>
<thead>
<tr>
<th>Location on any system:</th>
<th>Location on a 48K system:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest RAM memory address</td>
<td>49,151 ($FFFF)</td>
</tr>
<tr>
<td>800 (2000)</td>
<td>Relocated DOS, on completion of boot</td>
</tr>
<tr>
<td>7 bytes not used</td>
<td>80192 (9000)</td>
</tr>
<tr>
<td>1792 bytes of buffers of three files</td>
<td>80184 (9000)</td>
</tr>
<tr>
<td>355 ($253) bytes for input &amp; output</td>
<td>38400 (5000)</td>
</tr>
</tbody>
</table>

#### DOS Moves

- HIMEM here [Note 2]
- 16383 ($3FFF)
- 7424 ($1B00)
- 6912 ($1800)
- 2383 ($8FF)
- 2048 ($800)
- 1023 ($3FF)
- 512 ($200)

#### Lowest RAM Memory Address

- 0000 ($0000)

**Note 1:** This memory area is not affected when booting a Slave diskette. DOS is placed directly below the highest RAM Memory address that was available on the system that initialized the Slave diskette, whether appropriate to the present system or not.

#### B. Memory Areas Used by DOS and Either BASIC

<table>
<thead>
<tr>
<th>Without DOS, either BASIC sets HIMEM here [Note 2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Booting DOS sets HIMEM here [Note 2]</td>
</tr>
<tr>
<td>Highest RAM memory address: 49,151 ($FFFF) on a 48K system</td>
</tr>
<tr>
<td>10752 bytes (0A00) Disk Operating System</td>
</tr>
<tr>
<td>Integer BASIC program lines start at HIMEM and build down</td>
</tr>
<tr>
<td>Applesoft string start at HIMEM and build down</td>
</tr>
<tr>
<td>24576 (16000) High-resolution graphics, Page 2 [Note 5]</td>
</tr>
<tr>
<td>16384 (40000) High-resolution graphics, Page 1 [Note 5]</td>
</tr>
<tr>
<td>8192 (20000) FF (diskette) sets LOMEM at 12791 (53003) [Note 2]</td>
</tr>
<tr>
<td>2048 (10000) Diskette Applesoft (if used)</td>
</tr>
<tr>
<td>Either BASIC's variables occupy this space</td>
</tr>
<tr>
<td>start at LOMEM and build up [Note 4]</td>
</tr>
<tr>
<td>Applesoft program lines push LOMEM up</td>
</tr>
<tr>
<td>1023 ($3FF)</td>
</tr>
<tr>
<td>300 - 3FF Monitor and BASIC Workspace</td>
</tr>
<tr>
<td>300 - 3CF Vectors Reserved</td>
</tr>
</tbody>
</table>

**Note 2:** If your system is in Integer BASIC, the HIMEM pointer can be found (low byte first, then high byte) in locations 76-77 ($4C-$4D). If your system is in APPLESOFT BASIC, the HIMEM pointer is in locations...
113-116 (873-874), same format. See Table 2 for the value of HIMEM set by booting DOS. Increasing MAXFILES will move HIMEM down an additional 595 bytes for each file buffer added. For the locations of other Applesoft program pointers, consult your Applesoft II BASIC Programming Manual, Appendix I.

### Table 2: HIMEM Value Set by Booting DOS

When DOS is booted, HIMEM is set according to the amount of memory in the system:

<table>
<thead>
<tr>
<th>System size</th>
<th>Highest RAM address</th>
<th>HIMEM set by DOS boot</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Decimal</td>
<td>Hexadecimal</td>
</tr>
<tr>
<td>16K</td>
<td>16383</td>
<td>$3FFF</td>
</tr>
<tr>
<td>20K</td>
<td>20479</td>
<td>$4FFF</td>
</tr>
<tr>
<td>24K</td>
<td>24575</td>
<td>$5FFF</td>
</tr>
<tr>
<td>32K</td>
<td>32767</td>
<td>$7FFF</td>
</tr>
<tr>
<td>36K</td>
<td>36863</td>
<td>$8FFF</td>
</tr>
<tr>
<td>40K</td>
<td>49152</td>
<td>$9FFF</td>
</tr>
</tbody>
</table>

Note 3. The number -27136 could also be written 3640000, but Integer BASIC will not accept numbers greater than 32767. 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

\[-n - 65536\]  

Note 4. Using high-resolution graphics Page 1 erases the contents of memory locations 8192 through 16383. Unless DOS sets HIMEM to a value greater than 16383, an attempt to use high-resolution graphics Page 1 will erase part of DOS. This means that you cannot use Disk II and high-resolution graphics at the same time, unless your system contains at least 32K of memory.

If you are using diskette Applesoft, an attempt to use high-resolution graphics Page 1 will erase part of Applesoft. With diskette Applesoft, you may use high-resolution graphics Page 2, only, if your system contains at least 36K of memory. See Note 5.

Note 5. Using high-resolution graphics Page 2 erases the contents of memory locations 16384 through 24576. Unless DOS sets HIMEM to a value greater than 24575, an attempt to use high-resolution graphics Page 2 may erase part of DOS. This means that you cannot use Disk II and Page 2 high-resolution graphics at the same time, unless your system contains at least 36K of memory.
DOS ENTRY POINTS

Routine to re-connect DOS (if page 3 in over-written):

<table>
<thead>
<tr>
<th>System</th>
<th>Decimal address</th>
<th>Hexadecimal address</th>
</tr>
</thead>
<tbody>
<tr>
<td>48K</td>
<td>-25153</td>
<td>$DA1F</td>
</tr>
<tr>
<td>32K</td>
<td>23999</td>
<td>$DA1F</td>
</tr>
<tr>
<td>16K</td>
<td>7615</td>
<td>$DA1F</td>
</tr>
</tbody>
</table>

The Monitor command DEFL displays this number at the top right.

Locations containing the start address and length of a BLOADed program:

<table>
<thead>
<tr>
<th>System</th>
<th>Start address (low byte)</th>
<th>Program length (low byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Decimal</td>
<td>Hexadecimal</td>
</tr>
<tr>
<td>48K</td>
<td>43634</td>
<td>$AA72</td>
</tr>
<tr>
<td>32K</td>
<td>27250</td>
<td>$6A72</td>
</tr>
<tr>
<td>16K</td>
<td>19846</td>
<td>$2A72</td>
</tr>
</tbody>
</table>

To see the starting address or length after a BLOAD, type

PRINT PEEK(low byte) + PEEK(low byte (1))*256

Program to find the DOS locations containing the starting address and length of the most recently BLOADed program, on any size system:

```
2 REM BLOAD FINDER
7 H = 38400. REM DOS-BOOT HIMEM
8 I = 49152. REM HIGHEST ADDRESS
10 D$ = CHR$(4): REM CTRL-D
20 PRINT D$: "BSAVE FOO. A87777. L$777"
30 PRINT D$: "BLOAD FOO"
40 PRINT D$: "DELETE FOO"
50 FOR I = H + 1792 TO T
60 IF PEEK (I) > 119 OR PEEK (I + 1) > 119 THEN NEXT I
70 PRINT "LOCATIONS OF START"
80 PRINT "LOCATIONS OF LENGTH: "
90 FOR I = H + 1792 TO T
100 IF PEEK (I) < 119 OR PEEK (I + 1) < 0 THEN NEXT I
101 PRINT "LOCATIONS OF LENGTH: "
```

The values of H and I (lines 7 and 8) are shown for a 48K system. Appendix D, page 142, shows the correct values for your system. This program takes about 2 minutes to find the desired locations.

DOS character input and output routines:

See Chapter 19, especially Note 7 on page 195. For an example using the technique described, see the program on page 151.
The DOS commands are grouped into 5 categories in this appendix:

**Housekeeping commands**
- INIT
- RENAME
- VERIFY
- CATALOG
- DELETE
- MON
- SAVE
- LOCK
- NOMON
- LOAD
- UNLOCK
- MAXFILES
- RUN

**Access Commands**
- PP
- PR
- CHAIN
- INT
- IN

**Sequential Text File Commands**
- OPEN
- APPEND
- CLOSE
- POSITION
- READ
- EXEC
- WRITE

**Random-Access Text File Commands**
- OPEN
- READ
- CLOSE
- WRITE

**Machine Language File Commands**
- BSAVE
- BLOAD
- BRUN

Procedures used in DOS (including chancing in Applesoft) are summarized in Appendix E. The notation used in the summaries (and throughout the manual) is described in the following section.

**NOTATION**

Syntax refers to the structure of a computer command. A simple notation is used to describe the syntax of each DOS command.

Items in square brackets, [ ] and { }, are optional. These items are sometimes called parameters. Not all commands permit all parameters, but those parameters that are permitted in a given command may appear in any order, unless otherwise noted.

If a command uses a file name, the file name must come immediately after the command word itself; the first item following the command will be treated as a file name. The file name must be separated by a comma from any parameter that follows.

Curly brackets may be used to indicate when a certain key should be pressed:

(CTRL) hold down the key marked "CTRL" while another key is typed.
(CTRL)D means hold down the CTRL key while you type the letter D. Sometimes another notation is used: CTRL-D means the same as (CTRL)D.

(RETURN) press the key marked "RETURN". The (RETURN) required after every command is not shown.
(RESET) press the key marked "RESET".
(ESC) press the key marked "ESC".

CAPITAL letters and commas must be typed as shown. Lower case letters stand for items that you must supply.

f file name. This is from one to 30 characters. Any typeable character except the comma may appear in a file name. The first character must be a letter of the alphabet. For more details, see the next section.

Examples: CHESS
          RECIPE
          SUM OF SQUARES
          NEW45
          HOW-ABOUT-THIS

b another file name.
Example: SEPARATOR WITH LOW VELOCITY

v slot number. v specifies the Apple II slot in which the floppy controller card has been placed (usually slot 6).
  v initially defaults to the slot from which DOS was booted. It subsequently defaults to the last value specified for this parameter. v must be in the range 1 through 7.

Examples: 7

If v refers to a slot which does not contain a disk controller card, the system may stop and a program in memory may even be lost. See I/O ERROR, in Appendix B, for more details.

v volume number of a diskette. v initially defaults to the volume number of the diskette from which the system was booted. It subsequently defaults to the latest value specified for this parameter, or implicitly specified by a CATALOG command. v must be in the range 0 through 254.
Example: 101

Note: A diskette's volume number may not be 0. In a DOS command, specifying a volume number of 0 or simply v with no number is a "wild card" and tells the DOS to determine and use the volume number on the diskette.

D drive number (either 1 or 2). D initially defaults to one. It subsequently defaults to the latest value specified for this parameter.
Example: 2
FILE NAMES

File names may be up to 30 characters long, and must begin with a letter. The name cannot contain a comma, a RETURN, or a RETURN, which is used to terminate the command. Spaces that precede the first non-space character in a name are ignored. All name characters beyond the 30th are ignored.

When typing file name, the use of special keys such as ESC, the left-arrow and right-arrow keys, and certain keys typed with the CTRL key ("control" characters CTRL-C, CTRL-D) may have unexpected effects.

If a file name contains control characters, you won't see them printed, but they must be typed to use or delete the file.

The following AppleSoft program can be used to find any hidden characters except CTRL-\n\n(RIGHT), ESC, CTRL-H (left arrow) and CTRL-U (right arrow).

```
DATA 034 143 240 21 201 136
DATA 240 17 201 128 144 13
DATA 281 168 176 9 72 132
DATA 52 58 233 54 76 249
DATA 282 76 240 253
FOR I = 75S TO 756 + 27
READ V POKE I V NEXT I
POKE 34 0: POKE 35 3
CALL 19862
```

If you suspect you may have accidentally introduced a control character into a file name, type this program, SAVE it, and RUN it. The AppleSoft prompt ( ) will be displayed. Next type CATALOG

and you'll get a list of all the files, with any control characters shown as flashing characters. Control characters in program listings can also be found this way. To re-instate normal printouts, type PRF 0

HOUSEKEEPING COMMANDS

```
INIT f [\w] {\s} {\d}
```

Example: INIT HELLO, V18

The parameter v assigns a volume number to the diskette being initialized. Details on initializing diskettes are in Chapter 2 and Appendix E.
LOAD f [,Ss] [,Dd] [,Vv]

Example: LOAD DON JOHNS, V19, D1

Attempts to find Integer BASIC or Applesoft program file with name f on the diskette in the specified or default drive. If the volume numbers match and there is such a file, that program will be LOADED into the computer. It can then be LISTed, or RUN, or SAVED as with any program. LOAD closes any open text files, changes the Apple to the correct language for file f, and erases any program in memory before placing the new program in the Apple.

If file f is an Applesoft BASIC program, and Applesoft is not already in memory or available from the Applesoft firmware ROM card, the program Applesoft will be LOADED and RUN from the specified drive automatically, before file f is LOADED. If Applesoft is not on that diskette nor on the firmware ROM card, the message LANGUAGE NOT AVAILABLE will be displayed.

The instruction LOAD, without any parameters, will LOAD a program from cassette tape.

RUN f [,Ss] [,Dd] [,Vv]

Example: RUN ANNUITY, D2

LOADs file f from the specified or default drive (see the previous discussion), then also RUNs the program loaded. If just RUN is typed, the program in memory is RUN.

RENAME f, g [,Ss] [,Dd] [,Vv]

Example: RENAME SEPARATE, SEPARATE, S4, D1, V0

Finds the file named f on the diskette in the specified or default drive, and changes its name to g. The file's contents are not affected. If file f was open, it is closed.

RENAME does not check to see whether the file name g is already in use, so it is possible to use RENAME to put several files of the same name onto a diskette — a potentially confusing situation, at best.
DELETE [,$s] [,d] [,v]

Example: DELETE TEST

Removes the file named f from the diskette in the specified or default drive. If f was open, this command closes it. See Appendix C for more details of the deletion process.

If a file named f does not exist on the diskette, the message FILE NOT FOUND will result. To avoid this occurrence stopping your programs, first OPEN the file, then DELETE it.

LOCK f [,$s] [,d] [,v]

Example: LOCK LOVE LETTERS, V31

This command allows you to make file f, on the diskette in the specified or default drive, safe from accidental deletion or change. A LOCKed file is indicated in the CATALOG by an asterisk (*).

UNLOCK f [,$s] [,d] [,v]

Example: UNLOCK RECIPES, V31, D2

If you change your mind, and want to alter or remove a LOCKed file named f, on the diskette in the specified or default drive, this command allows such a change.

VERIFY f [,$s] [,d] [,v]

Example: VERIFY SAM

Performs a check that the information actually stored on the diskette in file f is self-consistent. (Technically, this is what happens: When the file is created -- with SAVE, SSAVE or WRITE -- DOS calculates a checksum byte for the contents of each output buffer and then stores that byte with the buffer's contents in a diskette sector. The VERIFY command calculates a new checksum byte for the actual contents of each file sector, and compares it with the checksum byte originally stored with that sector.) If a file VERIFYs, no message is given; it's safe to assume the information on the diskette has been stored correctly. If a file does not VERIFY, the message I/O ERROR is presented. You may VERIFY any type of file.

MON [C] 1, 1 [I, O]

Examples: MON 0
MON C, 1, 0

All disk commands and all information sent between the computer and the disk are normally not displayed on the screen. This command allows you to enable some or all of this display -- a helpful tool when debugging a program. If C is specified then disk commands are displayed. If I is specified then information being sent from the disk to the Apple, as Apple's input, will be displayed. If O is specified, then information being sent to the disk from the Apple, as Apple's output, will be displayed.

At least one of the three parameters must be present, or MON is ignored. The parameters may appear in any order, separated by commas. These parameters appear only in the commands MON and NOMON.

Note: MON remains in effect until a NOMON command, a change of language (FP or EN), a boot, or a restart (3D$G). Even RUNNING a program won't cancel a MON.

NOMON IGI 1,1 I.D1

Examples: NOMON C
          NOMON I, C

The NOMON command enables you to display disk commands and/or information sent between the computer and the disk: such information is normally not displayed on the screen. The NOMON command allows you to disable some or all of this display. The command NOMON C, I, O returns the system to its usual, default state.

If C is specified then disk commands are not displayed. If I is specified, then information being sent from the disk to the Apple, as Apple's input, will not be displayed. If O is specified, then information being sent to the disk from the Apple, as Apple's output, will not be displayed.

At least one of the three parameters must be present, or NOMON is ignored. The parameters may appear in any order, separated by commas. These parameters appear only in the commands MON and NOMON.

MAXFILES n

Example: MAXFILES 6

n is an integer from 1 to 16 that specifies the number of files that can be active at one time. When MAXFILES is executed, 595 bytes of memory (called a file buffer) are reserved for each file. When you boot the system, n defaults to 3, so that you will have 1785 bytes reserved for file buffers and will be allowed a maximum of 3 files open simultaneously.
All DOS commands except PRF, IN# and MAXFILES require a file buffer. Thus if you have MAXFILES 1, and one file is OPEN, an attempt to perform a DOS command (such as CATALOG) will cause the message NO BUFFERS AVAILABLE to be displayed.

Use of MAXFILES moves HIMEM. This can erase Integer BASIC programs or Applesoft strings. Use MAXFILES before loading and running a program. See the discussion in Chapters 5 and 7 if MAXFILES must be used from within a program.

ACCESS COMMANDS

FP [,Fs] [,Dd] [,Vv]
Example: FP, D2

This command puts your system into Applesoft BASIC. Any Integer BASIC or Applesoft program in memory is lost. If your computer contains the Applesoft firmware card, DOS uses that source for the language, regardless of the switch position on the card. If your system does not contain the Applesoft firmware card, DOS attempts to load and run the program named APPLESOFT on the diskette in the specified or default drive.

To place the APPLESOFT program onto a newly initialized diskette, first LOAD the APPLESOFT program from the Master Diskette, then (without RUNNING or LISTing the file) SAVE APPLESOFT on an initialized diskette. You must use the name APPLESOFT for this file.

Do not use RUN APPLESOFT to change languages. Everything looks fine at first, but DOS has not properly initialized the language. To avoid the resultant mess, always use FF.

INT
Example: INT

This command puts the Apple into Integer BASIC. Any Integer BASIC or Applesoft program in memory is lost.

CTRL-D ( also written (CTRL)D )
Example: 10 D$=CHR$(4)
          20 PRINT D$;"WRITE CHESS"

Every character PRINTed out by the Apple is first examined by DOS before it is sent on to the outside world. If the Apple PRINTs out a RETURN character (most PRINT statements automatically end with a RETURN), and the next character is a CTRL-D, this is a message to DOS that subsequent characters (until the next RETURN) are a DOS command. Most DOS commands may be used from inside a BASIC program. To do so, PRINT a string consisting of CTRL-D followed by the desired DOS command.

The recommended way to do this is to first create a string D$ consisting only of a CTRL-D, and then to use BASIC statements such as shown in the example. Note the use of CHR$(4) to create D$ (this works only in Applesoft, since the CHR$ function is not offered in Integer BASIC).

Instead, CTRL-D could have been typed inside quotation marks to create D$, but in this case no character is shown between the quotation marks.

Every character sent out by the Apple is first examined by DOS before it is passed on to the outside world. If the Apple sends out a RETURN character (most PRINT statements automatically end with a RETURN), and the next character is a CTRL-D, this is a signal to DOS that subsequent characters (until the next RETURN) are a DOS command. A DOS command from a program must appear in a PRINT statement whose first output character is CTRL-D and whose output is separated from preceding and from succeeding printed output by RETURN's. For additional information, see "Use of DOS from within a Program", in Appendix G.

PRF 8
Example: PRF 6

Sends subsequent Apple output to the device controlled from slot # 8, instead of to the TV screen. The command PRF 0 returns output to the TV screen. If the command is used from inside programs, it must appear as a PRINTed DOS command, as shown below:
10 D$=""; REM CTRL-D
20 PRINT D$; "PRF 1"

If no device controller card is installed in slot # 8, the system may "hang" and you'll have to press the RESET key to recover.

IN# 8
Example: IN# 6

Takes subsequent Apple input from the device controlled from slot # 8, instead of from the Apple keyboard. The command IN# 0 resets the normal keyboard input. If the command is used from inside programs, it must appear in a PRINTed DOS command, as shown below:
10 D$=""; REM CTRL-D
20 PRINT D$; "IN# 1"

If no device controller card is installed in slot # 8, the system may "hang" and you'll have to press the RESET key to recover.
CHAIN f [.Sa [.Dd] [.Vo]

Example: CHAIN PART TWO, D1, S7, V6

Used from within an integer BASIC program, it loads and runs the integer BASIC program named f on the diskette in the specified or default drive, but does not clear the value of any variables. This means that program f can operate on the results of the previous program, and can leave data for any following program. You cannot CHAIN Applesoft programs using this command; see the special procedure for Applesoft programs in Chapter 10 or Appendix C.

SEQUENTIAL TEXT FILE COMMANDS

OPEN f [.Sa [.Dd] [.Vo]

Example: OPEN SESAME, D2

Allocates a memory buffer of 595 bytes to the text file f, and prepares the system to write or read from the beginning of the file. This command is used with the WRITE and READ commands to create and retrieve sequential text files.

If there is no file f on the diskette in the specified or default drive, one is created. If a file named f is already OPEN, this command first CLOSEs that file, before OPENing the specified file.

CLOSE f

Example: CLOSE WINDOW

If you were WRITEing, a CLOSE causes all remaining characters in the output part of the file buffer to be sent to the diskette specified when that file was OPENed. CLOSE f deallocates the buffer associated with the sequential text file f. If CLOSE is used without a file name, all OPEN files will be closed, with the exception of the EXEC file. (There can only be one EXEC file OPEN at any time. When another is implicitly OPENed, the existing EXEC file, if any, is automatically closed)

If a program is interrupted by a CTRL-C or while a text file is OPEN, it's a good idea to type CLOSE to keep any data from being lost.

Files that have been allocated by an OPEN statement must be CLOSED. Failure to CLOSE a file that was OPENed and written to (by a WRITE) can result in loss of data.

WRITE f [.Bs]

Example: WRITE ADDRESS.DATA

After this command, PRINT statements send their output to the specified file instead of to the Apple's TV screen. With the Byte parameter, WRITE begins at the b-th byte of the file (see Chapter 6, page 69). WRITE is cancelled by the printing of any DOS command, or by an INPUT statement. The null DOS command (simply PRINTING a CTRL-D) will do.

WRITE must be issued in deferred-execution mode.

After this command all Apple output characters that would normally be displayed on the screen are sent to the diskette instead. This includes INPUT question-mark prompts, error messages, and other unwanted characters.

READ f [.Rb]

Example: READ SESAME

After this command, INPUT statements (and GET statements in Applesoft) obtain their response characters from the specified sequential text file instead of from the Apple's keyboard. With the Byte parameter, READING begins at the b-th byte of the file (see Chapter 6, page 69).

INPUT causes characters to be READ from the sequential file one field at a time. A field consists of from 1 to 32767 characters, ending with a RETURN character. However, because of the limited capacity of strings and input/output buffers, it is very difficult to store and retrieve fields of more than 255 characters.

READ is cancelled by the printing of any DOS command. A null DOS command (just PRINT a CTRL-D) will cancel READ. The READ command must be used in deferred-execution mode.

APPEND f [.Sa [.Dd] [.Vv]

Example: APPEND MORE INFO

This command opens the specified text file, but places the position-in-the-file pointer 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. An APPEND must be followed by a WRITE to the file of the same name. (APPEND must not be followed by OPEN, because OPEN will reset the position-in-the-file pointer back to the file's beginning.)
If you specify the value of the R parameter, a position-in-the-file pointer is placed at the beginning of the p-th field in the file, and EXEC will start executing from this point in the file.

As with POSITION, the R parameter used with EXEC should be thought of as the Relative-field position parameter. However, unlike POSITION, EXEC always counts fields from the beginning of your file, and p is always relative to 1. The other parameters work as usual.

If you specify the value of the R parameter beyond the end of the file you'll get an END OF DATA message.

**RANDOM-ACCESS TEXT FILE COMMANDS**

OPEN f, Lj [,Ss] [,d] [,w]v

Example: OPEN SESAME, L2

OPEN allocates a 395-byt file buffer to the random-access text file f, and sets the record length to the number of bytes specified by j. The number j must be in the range 1 to 32767; j defaults to 1.

OPEN is used with the READ and WRITE commands to create and retrieve random-access text files. Note that the L (Length) parameter is not optional: by definition, you must specify the record length of a random-access text file. Each time you use a particular random-access text file, you must OPEN the file with the same L parameter value. DOS then uses that value to calculate the starting position of any specified record.

If there is no file f, one is created.

CLOSE [f]

Example: CLOSE BOOK

If you were WRITING, a CLOSE causes all remaining characters in the output part of the file buffer to be sent to the diskette in the drive that was specified when the file was OPENed. CLOSE de-allocates the buffer associated with the random-access text file f. If CLOSE is used without a file name, all OPEN files will be closed, with the exception of an EXEC file, if any.

If a program is interrupted by a CTRL-C while a text file is OPEN, it's a good idea to type CLOSE to keep from losing data.


Example: EXEC UTILITY

Similar to RUN, except that f is a text (data) file containing BASIC and DOS commands as they would be issued from the keyboard. This allows you to set up files that can control the Apple, much as you would control the Apple yourself.

There can only be one EXEC command in effect at a time. If the EXEC file contains the immediate-execution command EXEC, the original EXEC file is closed and the new EXEC file is opened and executed. If EXEC has OPENed a file, the command CLOSE will not close the file being EXEC-ed. When an EXEC file has completed all its commands, it closes itself and stops. If a file being EXEC-ed contains a command to RUN any program, EXEC waits patiently until the program ends. Then the next command in the EXEC file is executed.

However, if a program is running while an EXEC file is open, any INPUT statement in the program will take the next field from the EXEC file as the response, ignoring the keyboard. Worse yet, if that response is an immediate-execution DOS command, the command will be executed before the program continues.

If you type CTRL-C to stop an Applesoft program that is running while an EXEC file is still open, the remaining commands in the EXEC file will usually not be executed.

POSITION f [,Rp]

Example: POSITION ADDRESS.DATA, R277

POSITION places the position-in-the-file pointer at the beginning of the p-th field following the one you're in. A field is a sequence of characters terminated by a RETURN. Subsequent READS or WRITES will proceed from that point in the file.

POSITION deals with a relative, not an absolute, position, since you count fields forward from where you are in the file when the POSITION is executed.

POSITION actually scans forward through the contents of the file, character by character, looking for the p-th RETURN character. It then places the position-in-the-file pointer at the first byte following the p-th RETURN character. If, in this search, it finds any byte in which no character has ever been stored, the message END OF DATA is given. Normally, this occurs when the p-th field ahead of the current position in the file is beyond the file's last entry.

POSITION actually scans forward through the contents of the file, character by character, looking for the p-th RETURN character. It then places the position-in-the-file pointer at the first byte following the p-th RETURN character. If, in this search, it finds any byte in which no character has ever been stored, the message END OF DATA is given. Normally, this occurs when the p-th field ahead of the current position in the file is beyond the file's last entry.
Files that have been allocated by an OPEN statement must be CLOSED. Failure to CLOSE a file that was OPENed and written to (by a WRITE) can result in loss of data.

WRITE f [], Rl [], Db

Example: WRITE ADDRESS, DATA, R3

After this statement, PRINT statements send their output to the specified file instead of to the Apple's TV screen. WRITE is cancelled by the printing of any DOS command, or by an INPUT command. The null DOS command (simply PRINTING a CTRL-D) will stop a WRITE with a minimum of effort. WRITE can be used only in deferred-execution PRINT statements.

The R (Record) parameter causes the WRITE to begin at the first byte of the r-th record, where each record contains the number of bytes, j, specified by the L parameter given with OPEN. r defaults to 0. If the R parameter is specified, the WRITE will begin at the b-th byte of the r-th record in the file.

After the WRITE statement, all Apple output characters that would normally be displayed on the screen are sent to the diskette instead. This includes INPUT question-mark prompts, error messages, and other unwanted characters.

READ f [], Rl [], Db

Example: READ SESAME, R3, Db

After this statement, INPUT statements (and GET statements in AppleLaunched) obtain their response characters from the specified random-access text file instead of from the Apple's keyboard. INPUT causes characters to be READ from the random-access file's current record, one field at a time.

A field can be from 1 to 32767 characters, ending with a RETURN character. However, no record should be more than j characters in length, where j is the record length specified when the file was OPENed.

The R (Record) parameter causes the READ to begin at the first byte of the r-th record, where each record contains the number of bytes, j, specified by the L parameter given with OPEN. r defaults to 0. If the R parameter is specified, the READ will begin at the b-th byte of the r-th record in the file.

READ is cancelled by the printing of any DOS command. A null DOS command (just PRINT a CTRL-D) will cancel READ.

MACHINE-LANGUAGE FILE COMMANDS

BSAVE f, Aa, Lj [], Db [], Wv

Examples: BSAVE PICTURE, A16384, L8192
BSAVE PICTURE, A54000, L52000

Creates a file named f, and stores the contents of a segment of the APPLE II's memory. The segment is specified by the starting address a, and the number of bytes to be stored j.

The examples shown above store a high-resolution picture, from the second high-resolution picture area. They are operationally identical; the second example just uses hexadecimal notation for the parameters.

BLOAD f, Aa, Ls, Dd, Wv

Examples: BLOAD PICTURE, A8192
BLOAD PICTURE, A52000

If a is not specified, then BLOAD places the specified file in Apple's memory beginning at the starting location of the memory area that was originally BSAVEd. If a is specified, then the data is placed in Apple's memory beginning at address a. Note that a machine-language program may no longer be executable if so moved.

Assume that a a high-resolution graphics picture has been BSAVEd on a diskette under the file name PICTURE. Then the first example shown above would place the picture into the first high-resolution picture area, which starts at memory location 8192 (decimal). The second example is equivalent: the address is shown in hexadecimal, as indicated by the "h" before the 3000.

Either example would clobber any version of Applesoft that is not in firmware.

BRUN f, Aa, Ls, Dd, Wv

Example: BRUN SUPER, A500A, V75

BLOADs the file f into Apple's memory beginning at location a. If the A parameter is omitted, the file is BLOADed starting at the same location from which it was BSAVEd. Once BLOADed, the file (which should be a machine-language program) is started by a machine-language jump (JMP) to location a.
APPENDIX G
SUMMARY OF DOS PROCEDURES

166 Booting DOS
166 Initializing a Diskette
166 Recovering from Accidental RESETs
166 Use of DOS from within a Program
167 Creating a Turbokey System
167 Creating and Retrieving Sequential Text Files
169 Adding Data to a Sequential Text File
169 Controlling the Apple via a Sequential Text File
170 Creating and Retrieving Random-Access Text Files
171 Copying a Text File
171 Chaining in Applesoft
This appendix contains summaries of the main procedures used in DOS. Each
procedure is listed on the preceding contents page, along with the page
number on which it appears.

**BOOTING DOS**

Replace "a" by the slot number in which the disk controller is located.

<table>
<thead>
<tr>
<th>Prompt</th>
<th>Character</th>
<th>Language</th>
<th>To boot DOS, type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>Integer</td>
<td>BASIC</td>
<td>PRs or INFs</td>
</tr>
<tr>
<td>)</td>
<td>Applesoft</td>
<td>PRs or INFs</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>Monitor</td>
<td>(CTRL)P</td>
<td></td>
</tr>
</tbody>
</table>

**INITIALIZING A DISKETTE**

To initialize a slave (memory dependent) diskette:

1) Boot DOS
2) Insert a blank diskette into the disk drive
3) Type in a greeting program, e.g.

```plaintext
10 PRINT "32K SLAVE DISKETTE INITIALIZED 5 MAY 88"
20 END
```
4) Assuming you choose to name the greeting program "HELLO",
type the command

```
INIT HELLO
```
5) After the IN USE light on the disk drive goes out, remove
the diskette and label it.

To make a master diskette, use the MASTER CREATE program, page 44.

**RECOVERING FROM ACCIDENTAL RESETS**

If DOS has been booted and then the RESET key is accidentally pressed, type
`300`
(that's the numeral zero after the D) to get back into the BASIC you left
with your program intact.

**USE OF DOS FROM WITHIN A PROGRAM**

DOS commands may be issued from within a program by PRINTing CTRL-D then
the command. First create a string $D$ which consists only of CTRL-D.

In Applesoft, $D$ may be created by the command

```
D$ = CHR(4)
```
since CTRL-D is the character whose ASCII code is 4.

In either BASIC, $D$ may be defined by typing

```
D$ = ""
```
then holding down the CTRL key while typing the letter D, and then typing
the closing quote. Control characters such as CTRL-D aren't displayed, so
what you'll see is

```
D$ = ""
```

This Applesoft program displays the CATALOG when RUN:

```plaintext
5 REM GREETING PROGRAM
40 $D$ = CHR$(45). REM CTRL-D
20 PRINT $D$ = "CATALOG"
```

Only one DOS command may be contained in a single PRINT statement. The
PRINT statement's quoted contents must begin with a CTRL-D and end with
the DOS command. The CTRL-D must be preceded by a RETURN (sent
automatically at the end of most PRINT statements).

These commands should only be used in deferred-execution mode (from within
a program), appearing after CTRL-D in a PRINT statement:

```
OPEN APPEND READ WRITE POSITION
```

The commands INIT and MAXFILES are best used only in immediate-execution
mode (not from within a program).

Other DOS commands may be used either in immediate-execution mode, or from
within a program where they appear after a CTRL-D in a PRINT statement.

**CREATING A TURNKEY SYSTEM**

To make a diskette that runs a certain program each time the diskette is
booted -- in the example we will use the program COLOR DEMO -- use the
following procedure:

1) Initialize a blank diskette, using the name HELLO for the
greeting program.
2) Place a disk containing the COLOR DEMOS program in drive,
and type

```
RUN COLOR DEMOS
```
Once you're satisfied the program runs correctly, return
to BASIC.
3) Put the newly initialized diskette into your drive and type

```
SAVE HELLO
to replace the old greeting program by the COLOR DEMOS
program.
```

**CREATING AND RETRIEVING SEQUENTIAL TEXT FILES**

When creating a sequential text file, an OPEN must precede a WRITE; once a
WRITE is executed, any subsequent PRINT commands send all characters to
the diskette. CLOSE the file when you're done. A WRITE command is
cancelled by an INPUT or the use of any DOS command in a PRINT statement
-- even just PRINTing CTRL-D will do.
This Applesoft program creates a sequential text file named SAMPLE whose first thirteen fields contain three strings and the integers 1 through 10:

5 REM MAKE SAMPLE
10 D$ = CHR$(40): REM CTRL-D
20 PRINT D$: "OPEN SAMPLE"
30 PRINT D$: "DELETE SAMPLE"
40 PRINT D$: "OPEN SAMPLE"
50 PRINT D$: "WRITE SAMPLE"
60 PRINT "HI HO": PRINT "HI HO"
70 PRINT "OFF TO THE DISK WE GO"
80 FOR J = 1 TO 10
90 PRINT J: NEXT J
110 PRINT D$: "CLOSE SAMPLE"

If you OPEN a file that already exists and then WRITE to it, you will overwrite part of the original file.

This Applesoft program retrieves the file SAMPLE described above, one field at a time. If you wish to see what is being READ from the disk, the command NOW I will cause input from the disk to be displayed.

5 REM RETRIEVE SAMPLE
10 D$ = CHR$(40): REM CHR$(40)
 IS CTRL-D
20 PRINT D$: "OPEN SAMPLE"
30 PRINT D$: "READ SAMPLE"
40 INPUT A$: BRK.C$: C
50 FOR I = 1 TO 10
60 INPUT W
70 NEXT I
80 PRINT D$: "CLOSE SAMPLE"

An OPEN must precede a READ. Once a READ is executed, any subsequent INPUT statements (in Applesoft, GETS also) obtain their response characters from the diskette instead of from the Apple’s keyboard. CLOSE the file when you’re done.

A READ is cancelled by PRINTing CTRL-D, whether or not it’s followed by a DOS command.

ADDING DATA TO A SEQUENTIAL TEXT FILE

This Applesoft program adds the two strings "TEST 1" and "AND NOW FOR TEST 2" to the end of a sequential text file called SAMPLE. Each string is in an additional field of the file.

5 REM APPEND SAMPLE
10 D$ = CHR$(40): REM CTRL-D
20 PRINT D$: "APPEND SAMPLE"
30 PRINT D$: "WRITE SAMPLE"
40 PRINT "TEST 1"
50 PRINT "AND NOW FOR TEST 2"
60 PRINT D$: "CLOSE SAMPLE"

CONTROLLING THE APPLE VIA A SEQUENTIAL TEXT FILE

When RUN, this Applesoft program creates a text file named DOIT containing the commands:

LIST 29,50
RUN HELLO
CATALOG

5 REM MAKE DOIT
10 D$ = CHR$(40): REM CTRL-D
20 PRINT D$: "OPEN DOIT"
30 PRINT D$: "WRITE DOIT"
40 PRINT "LIST 29,50"
50 PRINT "RUN HELLO"
60 PRINT "CATALOG"
70 PRINT D$: "CLOSE DOIT"

Once the text file DOIT is created, the command EXEC DOIT will cause the commands in the file DOIT to be executed one by one, just as if they’d been typed in from the keyboard.
CREATING AND RETRIEVING RANDOM-ACCESS TEXT FILES

This Applesoft program creates a random-access text file named RA-FILE, whose records are each 36 bytes long. Then it WRITEs the string "NAME ADDRESS" followed by the record number, into records 12 through 15 of the file. In lines 70 and 80, record number 13 is changed to contain the string "DOS".

2 REM MAKE RA-FILE
18 D4 = CHR$(4); REM CTRL-D
30 PRINT D4: "OPEN RA-FILE"
32 PRINT D4: "DELETE RA-FILE"
40 PRINT D4: "OPEN RA-FILE: L30"
50 FOR I = 12 TO 15
60 PRINT D4: "WRITE RA-FILE: R",I
70 PRINT "NAME ADDRESS":I
80 NEXT I
90 PRINT D4: "WRITE RA-FILE: R13"
100 PRINT D4: "DOS"
110 PRINT D4: "CLOSE RA-FILE"

This Applesoft program READS records 12 through 15 of the random-access text file RA-FILE. Note that you must specify each record before READING it in line 60. Line 60 examines the three leftmost characters of the input string AS, taken from each record. If those three characters are "DOS", the message "RECORD # HAS CHANGED" is PRINTed, and the search continues.

2 REM RETRIEVE RA-FILE
12 D4 = CHR$(4); REM CTRL-D
20 PRINT D4: "OPEN RA-FILE: L30"
30 FOR J = 12 TO 15
40 PRINT D4: "READ RA-FILE: R",J
50 INPUT A5
60 IF LEFT$(A5,3) = "DOS" THEN
70 PRINT "RECORD ",J," HAS CHANGED"
80 NEXT J
90 PRINT D4: "CLOSE RA-FILE"

COPYING A TEXT FILE

Moving a BASIC or a binary program file to another diskette is no problem; just LOAD or BLOAD the file's contents into the Apple, and then SAVE or BSAVE those contents back onto the other diskette. Remember to specify the starting address and length parameters when using the BSAVE command.

The COPY program allows you to bulk copy the contents of one diskette onto another. If any of the files on the original diskette are protected from copying, you'll get an error message:

1/0 ERROR
STOPPED AT 320

To copy a text file, or any unprotected file for that matter, use the FID program. The full instructions for using FID are in Appendix J.

CHAINING IN APPLESOFt

To RUN a series of Applesoft programs without erasing earlier values of variables and arrays use the following procedure.

Suppose you wish program PART ONE to chain to the program PART TWO. First, make sure the binary file CHAIN is on the same diskette with the program PART TWO. (If it isn't, use FID to copy it onto the right place.) Then simply insert these two lines as the last two lines to be executed in the PART ONE program:

PRINT CHR$(4): "LOAD CHAIN.520"
CALL 520"PART TWO"

No space or other character may be between the # and the " in the CALL command.
APPENDIX H

UPDATING DOS TO 16 SECTORS

174 How to Install the New Firmware
175 Multiple Disk Drive
176 Using DOS with the Language System
177 Setting up a TurboSystem with a Language Card
If you already have a Disk Operating System, and are using a version of
DOS that runs in 13 sectors, (DOS 3.2.1 or earlier), you will need to
change two proms on your disk controller card to update your system to 16
sectors. Any version of DOS earlier than release 3.3 will need to be
updated.

HOW TO INSTALL THE NEW PROMS

The new disk controller PROMs replace two PROMs on each Disk II controller
card in your Apple.

Inside the case, find the edge-connector "slots" at the rear of the main
board. In one or more slots you will find cards labeled 'Disk II
Interface Card' at the top edge. Make sure the power is off. Remove each
of these controller cards by grasping it at the top edge and gently
rocking it back and forth. Remove the ribbon cables from each card, so
you can work on it more easily.

Hold a controller card in one hand, IC side toward you, "fingers portion"
down. You will see four rows of two ICs on the card. Find the left IC in
the second row from the top. It should be marked 'P6' or '341-0010-xx'
(possibly under a copyright sticker), and the card below it should be
marked 'PROM P6'. This is the IC you will replace with the new PROM P6A.

Now find the left IC in the bottom row. It should be marked 'P5' or '341
0005-xx' (possibly under a copyright sticker), and the card below it
should be marked 'PROM P5'. This is the IC you will replace with the new
PROM P5A.

Now look at the new PROMs. The new PROM P6A should be marked 'P6A' or
'341-0028-xx', or both. The new PROM P5A should be marked 'P5A' or '341-
0027-xx', or both. You should be able to find these markings without
peeling off any stickers. If you can't find the numbers at all, see your
dealer.

Now that you know where everything is, you are ready to replace the old
PROMs with the new ones. Set the controller card down on a table and hold
it firmly. Using the IC puller, gently remove PROM P6 from its socket. You
may find a translator, marked 'Q3' on the card, blocking the IC puller. If
so, gently bend it out of the way until it is approximately vertical.
Grasp the IC with the puller, and gently rock it out of its socket.

Hold PROM P6A, marked 'P6A' or '341-0028-xx', so that it faces the same way
as the ICs on the card. The large rectangular notch should be on the left
side as you face the card. (There may be a small circular depression on
the other end. If you are not sure which end, ask your dealer.)
Making sure that each pin lines up with its hole, gently insert PROM P6A
into socket P6. If the pins are too widely spread to go in easily, you may
have to bend them in until they are vertical, by pushing them carefully
(gently) against a table top.

Now remove PROM P5 and replace it with PROM P5A, marked 'P5A' or '341-0027-
xx', just as you replaced P6 with P6A.

If you have more than two Disk II drives, you will need one controller card
for each two drives. Each of these cards will need the new PROMs. Install
these as you installed the first pair.

MULTIPLE DISK DRIVES

Now connect each Disk II drive to its controller card and plug the card
into its slot, in the order shown in the table below. The first drive
should go in the first location listed (slot 6, drive 1), and so forth,
until you have installed all your drives. You will be working downward
from slot 6, filling first the drive 1 and then the drive 2 connectors of
each card. This will ensure that the drives are in the most efficient
locations, and will increase the reliability of your system. Programs
will not try to use drives that are not there, or ignore drives that are there.

Placement of Disk II Drives

<table>
<thead>
<tr>
<th>Installation Order</th>
<th>Slot No.</th>
<th>Drive No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st drive</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>2nd drive</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>3rd drive</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>4th drive</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5th drive</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>6th drive</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

Label each drive 'DRIVER 1' or 'DRIVER 2', as appropriate.
Check that all connections are secure and that everything has been installed in its proper place. PROM #5 of each controller card should have been replaced by PROM 5A (341-0027-xx). PROM #6 of each controller card should have been replaced by PROM 6A (341-0028-xx). A Disk II ribbon cable should be plugged into the drive 1 connector of each Disk II controller card. Slot 6 should have a controller card.

When you are satisfied that everything is installed properly, you can replace the Apple's cover and go on to update your existing 13 sector diskettes to 16 sectors, using MUPTR.

**USING DOS WITH THE LANGUAGE SYSTEM**

You can use DOS and both Applesoft and Integer BASIC with the Apple Language System installed. You do not need the BASICS Diskette to do this. To begin a 16 sector DOS after using any language on the Language System which uses the Pascal Operating System, follow these steps:

1) Be sure to save any program you have been working on. Otherwise, it will be erased from memory when you begin DOS.

2) Get into the Command mode of the Pascal Operating System (so that you can see the line which begins with "Command: ").

3) Replace the diskette in your Drive 1 (connected to the controller card in your highest numbered slot) with the DOS System Master diskette.

4) Type H (for Halt). This will cause that disk drive to boot your System Master diskette.

Your System Master diskette will now load either Applesoft or Integer BASIC -- whichever is NOT already available on your main board -- into the storage space available on your Language Card. If you have a standard Apple II, the System Master, when booted, will load Applesoft into your Language Card. If you have an Apple II Plus, Integer BASIC will be loaded. In either case, a message will appear, telling you which BASIC is being loaded during the booting of your System Master. All of the DOS commands will now work in the using and writing of programs in both Applesoft and Integer BASIC.

To learn the procedure for starting DOS after using a language which does NOT use the Pascal Operating System, see the manual for that language.

With the Language System installed, you can boot DOS upon turning on your Apple, with the System Master diskette in Drive 1.

To return to another Language System language from DOS, replace the System Master in Drive 1 with the startup diskette for that language. Then type PR#6 and press RETURN.

**SETTING UP A TURNKEY SYSTEM**

When you insert a DOS 3-3 diskette in Drive 1 and turn on the power, your Apple will run the HELLO program on that diskette, if it has in ROM the BASIC in which the HELLO program is written. That is, an Apple II will run an Integer BASIC HELLO program, an Apple II Plus will run an Applesoft HELLO program, and an Apple II with the Integer BASIC Card will run a HELLO program in either BASIC. If you don't have that BASIC in ROM, you will get a LANGUAGE NOT AVAILABLE error message and the prompt for the BASIC it does have.

If you have the Apple Language System, things get a bit more complicated. Without further ado, you can run programs in the BASIC in ROM, but before you can use the other BASIC, you must load it into the Language Card. The simplest way to do this is to boot the DOS 3-3 System Master Diskette, which is provided with two programs, HELLO and APPLESOFT, that together make sure that both BASICS are loaded when you boot DOS. HELLO, written in Applesoft for the Apple II Plus, loads Integer BASIC. APPLESOFT, written in Integer BASIC for the Apple II, loads Applesoft. Your machine will run the appropriate program, and afterwards behave as if it had both BASICS in ROM.

You can now insert any other DOS 3-3 diskette, and run any program on it.

If you wish to make a diskette that will boot on all systems, and load the Language Card if necessary, you can do it by copying the appropriate files onto it. Here's how:

1. Boot the DOS 3-3 System Master Diskette.

2. LOAD HELLO from the System Master, then insert a blank diskette and type

   INIT HELLO

3. When INIT is finished, reinsert the System Master, BRUN MASTER CREATE and type

   HELLO

   when asked for the name of the greeting program. Insert the blank diskette when told to do so by MASTER CREATE.

4. Using PID, copy these files onto your newly initialized diskette:

   APPLESOFT

   INTBASIC

   FPBASIC

5. Put any programs and other files you want on this diskette. Make sure you have at least the following files:

   HELLO

   APPLESOFT

   INTBASIC

   FPBASIC

When you boot this diskette on an Apple with the Language Card, it will load the Language Card with the BASIC not in ROM, so that the Apple can run any program in either BASIC.
If you have a turnkey program you want the Apple to run when it is turned on, you can do it by changing the HELLO and APPLESOFT programs so that each of them will run your turnkey program (we’ll call it TURNKEY here). You will find it easier to edit these programs if you first RUN LOADAPA, then use the ↓↓ command, so that you can see the CTRL-D in each embedded DOS command.

Start with the diskette you have just made in steps 1 to 5, and do the following:

6. UNLOCK HELLO
7. LOAD HELLO
8. LIST it and look at the last few lines
   240 END
   250 REM
   260 REM --NO CARD OR CAN’T RELOAD
   270 REM
   280 IF PEEK (768) = 0 THEN END
   290 PRINT: PRINT "...LANGUAGE CARD CANNOT BE RELOADED":
   300 PRINT "...UNTIL THE SYSTEM IS REBOOTED...
   310 END

On the screen, the lines will be wrapped at 40 characters; here they are broken elsewhere for clarity.

9. Change these lines so they read as follows:
   * 240 GOTO 300
   250 REM
   260 REM --NO CARD OR CAN’T RELOAD
   270 REM
   * 280 IF PEEK (768) = 0 THEN 300
   290 PRINT: PRINT "...LANGUAGE CARD CANNOT BE RELOADED":
   300 PRINT "...UNTIL THE SYSTEM IS REBOOTED...
   * 310 PRINT "CTRL-DUN TURNKEY"

The lines changed or added are marked with asterisks. The lower case
CTRL-d in line 310 represents a CTRL-D which won’t show up on the
screen or in a listing. If you don’t like invisible control characters
in your files, you can declare a string D$ containing a CTRL-D, which
would make line 310 look like this (in Applesoft):
   * 310 D$=CHR$(4) : PRINT D$;"RUN TURNKEY"
If you do this, make sure you declare D$ before you use it.

You may also wish to change or delete some of the PRINT statements
in the program, but make sure you leave the ones with DOS commands (these
will have CTRL-Ds in them).

10. SAVE HELLO
11. LOCK HELLO

12. UNLOCK APPLESOFT, LOAD it, and LIST it, looking at the last few
    lines.
    250 GOTO 310
    260 REM
    270 REM --NO CARD OR CAN’T RELOAD
    280 REM
    290 IF PEEK (768) = 0 THEN 310
    300 PRINT "...LANGUAGE CARD CANNOT BE RELOADED":
    320 PRINT "...UNTIL THE SYSTEM IS REBOOTED...
    310 PRINT "CTRL-DUN TURNKEY"

13. Change these lines to read as follows:
    250 GOTO 310
    260 REM
    270 REM --NO CARD OR CAN’T RELOAD
    280 REM
    * 290 IF PEEK (768) = 0 THEN 310
    300 PRINT "...LANGUAGE CARD CANNOT BE RELOADED":
    320 PRINT "...UNTIL THE SYSTEM IS REBOOTED...
    * 310 PRINT "CTRL-DUN TURNKEY"

Line 310 can also be put in the form
   * 310 D$="CTRL-D";PRINT D$;"RUN TURNKEY"
to make the CTRL-D easier to find.

14. SAVE APPLESOFT
15. LOCK APPLESOFT

16. RENAME your turnkey program TURNKEY. A sample TURNKEY program, in
    Applesoft, is below:
    100 TEXT : HOME
    110 VIAB 3: PRINT "DOS TOOL KIT " ; TAB( 30) ;"30 NOV 1979"
    120 VIAB 5: PRINT "(C) COPYRIGHT 1979, APPLE COMPUTER INC."
    130 FOR I = 1 TO 1500: NEXT I
    140 PRINT "CTRL-DUN RIBBIT"

17. Make sure your diskette has the following files on it:
    HELLO
    APPLESOFT
    INTBASIC
    FFRIASC
    TURNKEY

18. Now test your turnkey diskette by putting it in drive 1 and
turning the power off and on. It should boot DOS 3.3, then load
Integer BASIC or Applesoft if necessary, then run the program
named TURNKEY.
APPENDIX I

USING THE BASICS DISKETTE

182 Booting Diskettes by using "BASICS"
The "BASICS" diskette that came with the DOS system master can be used to boot 13-sector diskettes on your 16-sector system. Versions of DOS earlier than 3.3 are 13-sector DOS.

Apples come in two flavors: Apple II and Apple II Plus. The Apple II has Integer BASIC in Read-Only Memory (ROM) on the main printed-circuit board. The Apple II Plus has Applesoft BASIC in ROM on the main board. The language in ROM is the language the Apple will come up in when it is turned on, if it has no disk drives or accessory cards.

BOOTING DISKETTES USING "BASICS"

To use either Integer or Applesoft BASIC, insert the "BASICS" diskette into the drive, and turn the Apple on. In about five seconds the screen will say

```
INSERT YOUR 13-SECTOR DISKETTE
AND PRESS RETURN
```

Insert any 13 sector DOS/BASIC diskette and press RETURN. The Apple will now behave as described in this Manual, except that one step is added to the boot (startup) procedure:

When you turn the Apple on, or type PR#6, IN#6, C6#6, or 6CTRL-F, the "BASICS" diskette must be in the drive, each time you want to boot a 13 sector DOS.

When the cue comes on the screen, insert the 13 sector DOS/BASIC diskette you wish to boot. If you are in either BASIC and you type PR#6 or IN#6, you will have to insert "BASICS", then rela"sert your previous diskette.

Once you have DOS running, you can switch from one BASIC to the other by using the INT and FP commands normally.

As always, you can tell which BASIC you are in by the prompt you see on the screen: > for Integer BASIC and $ for Applesoft BASIC.

A diskette designed for "turnkey" operation (that is, one that starts executing a program when booted) can be run as before, after the "BASICS" diskette has been run.

You can reboot DOS without turning the power off. Simply insert "BASICS", type PR#6 or IN#6 and press RETURN, then at the cue insert the diskette you wish to boot.
FID (for File Developer) extends the abilities of 16-sector DOS in two ways. First, it lets you easily catalog, copy, delete, lock, and unlock all types of DOS files. Second, it lets you copy from one diskette to another with only one disk drive.

FID runs on any Apple with Applesoft, 32K or more of memory, and one or more Disk II drives. The FID destination diskette (the one you will be copying to) must be INITIALIZED in order for FID to work.

STARTING UP

To use FID, boot the DOS System Master Diskette, then type

`RUN FID`

You will see a list of commands on the screen. Each of them can be executed by typing the number in angle brackets `< >`.

FILENAMEs AND WILDCARD CHARACTERS

Some of the commands ask you for a filename. The filename is typed just as for a DOS command, except that you can replace part of the filename with the "wildcard" character (`*`). For example, if you typed the filename

`FR=CD`

the program would select all the files whose names began with ‘FR’ and ended with ‘CD’. If you typed

`=N=
```

all files whose names contained ‘N’ would be selected. If you typed

```

all files on the diskette would be selected. ("On which diskette?", you might ask. This will be explained below.)

If you used a wildcard character in a filename specification, you will be asked

**DO YOU WANT PROMPTING?**

If you reply by typing

`Y`

for Yes, you will be asked to verify each filename before it is acted on.

If you want the file, when the system displays it on the screen, type

`Y`

If you don’t want the file, type

`N`
Note: The Reset Slot and Drive command should not be confused with the Reset key. They will always be capitalized differently, to avoid confusion.

LOCK FILE

To lock a file, type 5
and, when prompted, type the filename. If you type an invalid filename, you will be prompted until you give a valid one.

You will now be asked to insert the diskette containing this file, and asked whether you wish to ESCape or proceed. If you proceed and the file is on the diskette inserted, it will be locked. When you catalog the diskette, the filename will be preceded by an asterisk *, indicating that the file is locked. If the file is not on the diskette, you will get a NO FILES SELECTED error message; you can recover by pressing any key except Reset, Shift, Ctrl, or Esc.

UNLOCK FILE

To unlock a file, type 4
and, when prompted, type the filename. If you type an invalid filename, you will be prompted until you give a valid one.

You will now be asked to insert the diskette containing this file, and asked whether you wish to ESCape or proceed. If you proceed and the file is on the diskette, it will be unlocked. When you catalog the diskette, the filename will not be preceded by an asterisk *, indicating that the file is not locked. If the file is not on the diskette, you will get a NO FILES SELECTED error message; you can recover by pressing any key except Reset, Shift, Ctrl, or Esc.

VERIFY

The Verify command is the same as the DOS VERIFY command, and handles all types of files. To use it, type 8
and answer the prompts. When a file has been verified, the screen will display its name and the word DONE.

If a file can't be read and is thus invalid, you will get an I/O ERROR message, and the program will prompt you to press a key. When you do so, you will return to the main menu.

COPY FILE

FID can copy files on either a multiple-drive or a single-drive system. To copy files, type C
and you will be prompted for Source slot and drive numbers, then Destination slot and drive numbers.

Multiple-Drive Copying

If the slot and drive numbers are different, you will be asked for a filename, which can contain wildcard characters. You can copy a whole diskette by typing * in response to this prompt. After you have given the filename, you will be asked to insert the appropriate diskettes and press Enter, to start the copy, or any key but Reset, Shift, Ctrl, or Esc, to go ahead. If the files you asked for are on the source diskette, and if there is enough room for them on the destination diskette, then you will be informed that your file has reached its new home; otherwise, you will get a DOS error message.

If the destination diskette already contains a file with the name you used, you will get the message

FILE ALREADY EXISTS.
TYPE IN NEW FILE NAME FOR THE COPY OR
<RETURN> TO REPLACE PRESENT FILE OR
<CTRL-C><RETURN> TO CANCEL COPY

If you enter a new filename, the file will be transferred and stored under that new name. If you press RETURN, the old file will be replaced by the new one, if the old file is not locked. If it is, you will see

FILE LOCKED.
DO YOU WANT TO REPLACE IT ANYWAY?

If you type Y
the file will be replaced; if you type N
you will get this message again:

FILE ALREADY EXISTS.
TYPE IN NEW FILE NAME FOR THE COPY OR
<RETURN> TO REPLACE PRESENT FILE OR
<CTRL-C><RETURN> TO CANCEL COPY

If, when you see this message, you press CTRL-C and then RETURN, the file will not be copied. If no files were found to copy, the program will print

NO FILES SELECTED

If a file was found, the program will print

DONE

Then you will be prompted:

PRESS ANY KEY TO CONTINUE

Pressing a key will return you to the menu.

Single-Drive Copying

If the specified slots and drives for the source and destination are the same, the program will tell you when to put the source and when to put the destination disk into the drive. Thus, copies can be made between two disks with only one drive.

DELETE FILE

This command deletes a file or set of files from the default diskette.

To use it, type

6

You will be asked

FILENAME?

When you type the filename, the program will try to delete the named file. If all goes well, you will see the message

<DONE>

on the screen. If the file could not be found, you will get the message

NO FILES SELECTED

If the file was locked, you will get the message

FILE LOCKED

and the file won’t be deleted.

To return to the menu, press a key other than RETURN, ESC, CTRL, SHIFT, or RESET when you see the prompt

PRESS ANY KEY TO CONTINUE

SPACE ON DISKETTE

This command lets you find out how many sectors are used, and unused, on the default diskette. To use it, type

1

and the numbers will be displayed on the screen.

QUIT

This command lets you exit the program cleanly. When you type

9

you will see the same prompt on the screen as you saw before you ran PID.

ERROR HANDLING

If the program detects an irrecoverable error, it will display an error message and abort the operation. If a Copy operation is aborted, you should catalog the destination diskette and delete any spurious file that may have been created by the operation. This spurious file, if it exists, will have the name of the original file, or a new filename if you gave it one, but it will be shorter than the original file, as it will be incomplete.

Error messages are generated for these errors: DISK FULL, DISK WRITE PROTECTED, FILE LOCKED and I/O ERROR. If one of these occurs, PID will wait for you to press a key, then return you to the main menu.

If a DOS error other than one of these occurs, its error code number will be printed. This will happen only if you clobbered DOS, or clobbered PID, or tried to copy a BAD FILE. If any of these things happened, you should reboot DOS, and rERUN PID.

If an error causes a spurious file to be placed on your destination diskette, delete that file immediately, before copying any other files to the destination diskette.
The current DOS is a "16 sector" DOS. Earlier versions of DOS are 13 sector. For a discussion of sectors, see the section entitled "Overview of the Storage Process" in Appendix C.

A 16 sector DOS will not work with diskettes that have been INITIALIZED with a 13 sector DOS. This is because the information is arranged differently on a 13 sector diskette. If you try to use 13 sector diskettes with a 16 sector DOS, you'll see the message UNABLE TO READ/WRITE.

STARTING UP

The MUFFIN program converts 13 sector diskettes, and the programs they contain, to 16 sector. It does this by rearranging the information and then writing it onto another diskette. The diskette it writes the information onto (the "destination") must have been INITIALIZED with a 16 sector DOS. After the information from the old diskette (the "source") is converted onto the destination, the 13 sector version of it will still exist on the source diskette.

To maximize the amount of disk space which is usable by this DOS, it's best to convert all your 13 sector files to a 16 sector diskette and then re-INITIALIZE the old diskettes with the current DOS.

The MUFFIN program will work with either one or two disk drives.

This example shows how to use MUFFIN to convert files from a 13 sector diskette onto a 16 sector diskette, on an Apple with one disk drive. It assumes your disk drive is connected to the controller card in slot #6.

1) Place the System Master diskette in your drive and type
   BRUN MUFFIN

2) When the message

   ****************************
   * APPLE II DOS 3.2 TO 3.3 CONVERTER *
   * MUFFIN VERSION 0 *
   * COPYRIGHT 1979 APPLE COMPUTER INC. *
   ****************************
   CHOOSE ONE OF THE FOLLOWING OPTIONS
   (1) CONVERT FILES
   (2) QUIT
   WHICH WOULD YOU LIKE?

appears, type

1

to indicate that you wish to convert files.

3) Now you'll see the question
   SOURCE SLOT?
   Type 6 (the number of the slot your disk controller card is in). To the
   next question: DRIVE? respond with 1.

4) When you see
   DESTINATION SLOT?
   type 6, and then answer the question DRIVE? by typing 1.

5) Now you'll be asked for the name of the file to be converted:
   FILENAME?
   Type - (an equals sign) here. The - is a symbol which represents the
   names of all the source diskette's files. This response means that you
   wish to convert the entire contents of the source diskette.

6) Before anything will be converted, you'll see
   DO YOU WANT PROMPTING?
   For now, type N for No.

7) When you see
   INSERT DISKS THEN PRESS <ESC> TO RETURN TO
   MAIN MENU OR ANY OTHER KEY TO BEGIN
   remove the System Master diskette from the drive and insert a 13 sector
   diskette containing files you wish to convert to 16 sector. Then press
   RETURN.

8) After the message
   INSERT SOURCE DISK AND PRESS A KEY
   appears, simply press RETURN.

9) The program will now find the first file on your 13 sector diskette
   and print its name. Then it will stop and wait for you to replace that
   diskette with a 16 sector (INITIALIZED) diskette. Do this and press
   RETURN.
   You'll see
   DONE
   when the file has been converted. Then you'll be instructed to re-insert
   the source diskette. Repeat this procedure until all your 13 sector
   files have been converted into 16 sector files.

NOTE: To convert large files, you may need to swap diskettes several times
for to get whole file transferred.

When you are converting files using more than one disk drive, specify the
slot and drive numbers of the source and destination diskettes when the
program asks for them (steps 3 and 4 above). Place the diskettes in the
appropriate drive (step 8) before the conversion begins.

When you see the message
   INSERT DISKS 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 here, the program will stop in its tracks and send you back to
   the menu.
If you try to convert a 13 sector file with the same name as a file already on the destination diskette, you'll see:

FILE [filename]
ALREADY EXISTS.
TYPE IN A NEW FILE NAME FOR THE COPY
OR <RETURN> TO REPLACE EXISTING FILE
OR <CTRL-C><RETURN> TO CANCEL COPY:

At this point you may either (a) type a new name for the file to be converted, (b) convert the 13 sector file and have it replace the file currently on the 16 sector destination under that name or (c) type CTRL-C and press RETURN to halt the conversion.

USING WILDCARD CHARACTERS

As you saw in the example, * may be used to mean "all files on the diskette". The * may also stand for any character or group of characters within a filename. For example, if you respond to the question FILENAME?
by typing
FI=*E
you will convert all files on the 13 sector source diskette whose names begin with FI and end with LE. In the same way, *.TXT converts all files whose names end with .TXT and --*, all files with names containing an *.

Because the * acts like a wild card in a card game it's called the "wildcard character".

The question
DO YOU WANT PROMPTING?
will appear when you use the wildcard character. If you respond with Y here, the program will stop after finding each file on the 13 sector diskette and wait for you to indicate whether or not to convert that file (by typing Y or N, or Q to stop and return to the menu). A response of N to
DO YOU WANT PROMPTING?
indicates that all the files in the wildcard group should be converted.
machine-language files 92-93
MAKE TEXT 61-66
MASTER CREATE 44-46
master diskettes 44-46
MAXFILES 31, 43-44, 135-156
memory requirements 13, 140
memory usage and map 140
MON 42-43, 155
Monitor mode $50
Monitor I/O registers 191, 103
MUFFIN program 192-194

NO BUFFERS AVAILABLE 121
NO MON 42-43, 155
NOT DIRECT COMMAND 122
notation 140-145

D (output) parameter
with NON 42
with NONON 42
ONERR GOTO 114
OPEN 49-71, 136, 161
output registers
DOS 191, 103
Monitor 191, 103
overwriting 53-54, 69

P; see relative-field position
(R) parameter
P, control $03
POSITION 66-68, 125, 159-160
PR# 11, 109-112, 157
PRINT (with CURD=0) 29-31, 50
PROGRAM TOO LARGE 29
prompt characters 11, 36
PRINT 174-175

quotations marks 39

R parameter
with EXEC 79
with POSITION 67-69, 125
with READ 67-68, 150
with WRITE 67-68, 150
RMT subroutine 94-98
random-access text file 82-89
creating and retrieving 82-85,
178 differences from sequential 82
sample programs 82-67
RANGE ERROR 116
READ
with random-access text files
88-89, 157, 162
with sequential text files
49-71, 136, 159
read or write a track
or sector 94-98
record 82, 126
record number (R) parameter 82-86
relative-field position
(R) parameter 67-69
RNAME 17, 153
RESET 11, 18, 169, 166
RETRIEVE TEXT 65-66
RETURN 11, 12, 149
RETURN character 51
ribbon cable 2-4
RUN 25-26, 153

S: see S (slot) parameter
S (slot) parameter 22, 149
SAVE 13, 25-26, 152
schematics 143-146
sector 16, 94, 124
sector allocation order 135
13 to 16 conversion 192-194
format 127-128
sequential text files 49-71
creating and retrieving 49-71,
167-168
EXEC 76, 169
sample programs 49-71

slave diskettes 13, 141
slots 3
slot (S) parameter 22, 149
syntax 140-150
SYNTAX ERROR 128
System Monitor diskettes 10, 13-14

T

text file, random-access:
see random-access text file
text file, sequential:
see sequential text file

TRACE 44
track allocation order 135
track bit map 124, 133-134
track/sector list 124, 128-129
tracks 94, 124
turnkey system 34-35, 167, 177

U

UNLOCK 35, 154
unpacking 2

V

V: see V (volume) parameter
V (volume) parameter 23, 149
VERIFY 35-36, 154
VOLUME HIGHEST 118-119
volume number: see volume parameter
volume (V) parameter 23, 149
VTOC (volume table of contents)
132-133

W

wildcard characters 184, 193
WRITE
with random-access text files
88-89, 126, 162
with sequential text files
49-71, 124-125, 139
write protecting 36-37
WRITE PROTECTED 116

X

y-register 94-95
## DOS MESSAGE INDEX

<table>
<thead>
<tr>
<th>ERROR #</th>
<th>MESSAGE</th>
<th>PAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>END OF DATA</td>
<td>53, 67, 120</td>
</tr>
<tr>
<td>10</td>
<td>FILE LOCKED</td>
<td>35, 67, 120</td>
</tr>
<tr>
<td>6</td>
<td>FILE NOT FOUND</td>
<td>17, 18, 25, 36, 46, 59, 118, 154</td>
</tr>
<tr>
<td>13</td>
<td>FILE TYPE MISMATCH</td>
<td>35, 46, 121, 152</td>
</tr>
<tr>
<td>8</td>
<td>I/O ERROR</td>
<td>22, 26, 36, 119, 154</td>
</tr>
<tr>
<td>1</td>
<td>LANGUAGE NOT AVAILABLE</td>
<td>28, 115, 153</td>
</tr>
<tr>
<td>12</td>
<td>NO BUFFERS AVAILABLE</td>
<td>43, 121, 156</td>
</tr>
<tr>
<td>15</td>
<td>NOT DIRECT COMMAND</td>
<td>48, 122</td>
</tr>
<tr>
<td>14</td>
<td>PROGRAM TOO LARGE</td>
<td>26, 122</td>
</tr>
<tr>
<td>2, 3</td>
<td>RANGE ERROR</td>
<td>92, 116</td>
</tr>
<tr>
<td>11</td>
<td>SYNTAX ERROR</td>
<td>18, 26, 28, 79, 92, 120</td>
</tr>
<tr>
<td>7</td>
<td>VOLUME MISMATCH</td>
<td>23, 118-119</td>
</tr>
<tr>
<td>4</td>
<td>WRITE PROTECTED</td>
<td>37, 116-117</td>
</tr>
</tbody>
</table>

## PROGRAM INDEX

<table>
<thead>
<tr>
<th>PROGRAM</th>
<th>PAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greeting (HELLO)</td>
<td>13-14</td>
</tr>
<tr>
<td>COUNT</td>
<td>15</td>
</tr>
<tr>
<td>CATALOG-Greeting</td>
<td>29-30</td>
</tr>
<tr>
<td>COLOR DEMO</td>
<td>34-35</td>
</tr>
<tr>
<td>ANIMALS</td>
<td>37</td>
</tr>
<tr>
<td>COPY</td>
<td>39-40</td>
</tr>
<tr>
<td>MASTER CREATE</td>
<td>44-46</td>
</tr>
<tr>
<td>EXEC DEMO</td>
<td>74-75</td>
</tr>
<tr>
<td>CAPTURE</td>
<td>75-77</td>
</tr>
<tr>
<td>RANDOM, APPLE PROMS</td>
<td>86-88</td>
</tr>
<tr>
<td>CHAIN</td>
<td>186-187</td>
</tr>
<tr>
<td>PID</td>
<td>184-189</td>
</tr>
<tr>
<td>MUFFIN</td>
<td>192-194</td>
</tr>
</tbody>
</table>
**DOS QUICK REFERENCE CARD**

On this card, DOS commands are grouped into three categories:

- **Housekeeping commands**:
  - `INIT`, `LOAD`, `DELETE`, `VERIFY`, `MAXFILES`,
  - `CATALOG`, `RUN`, `LOCK`, `MON`.
  - `SAVE`, `RENAME`, `UNLOCK`, `NONON`.

- **Access Commands**:
  - `PP`, `INT`, `PR#`, `IN#`, `CHAIN`.

- **Sequential Text File Commands**:
  - `OPEN`, `READ`, `APPEND`, `EXEC`.
  - `CLOSE`, `WRITE`, `POSITION`.

- **Random-Access Text File Commands**:
  - `OPEN`, `CLOSE`, `READ`, `WRITE`.

- **Machine-Language File Commands**:
  - `BLOAD`, `BHUN`, `BSAVE`.

**NOTATION AND SYNTAX**

A “parameter” is a capital letter, usually followed by a number (shown here by a lower-case letter), which gives additional information for executing a command. Multiple parameters may appear in any order, but must be separated from each other by a comma. A parameter shown in square brackets (like this) is optional.

A file name (shown here by X) must immediately follow its command word. File names must begin with a letter; only the first 30 characters are used. A comma separates a file name from a following parameter.

CTRL-D (type D while holding down CTRL key) is used in PRINT statements to indicate the start of a deferred-execution DOS command.

Integer BASIC example:

```
10 D$ = "" : REM "CTRL-D"
20 PRINT D$ ; "CATALOG"
```

AppleSoft BASIC example:

```
10 D$ = CHR$(4) : REM CTRL-D
20 PRINT D$ ; "CATALOG"
```

The term "BASIC" alone is used to mean either Integer BASIC or AppleSoft BASIC. The term "file" alone means any type of diskette file.

**COMMAND PARAMETERS**

An error message is given if a DOS command quantity is too large or too small.

**ALL FILES**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>As shown</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slot</td>
<td>.Sx</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>Drive</td>
<td>.Dx</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Volume</td>
<td>.Vv</td>
<td>0</td>
<td>254</td>
</tr>
</tbody>
</table>

* Using V0 is like omitting the Vv parameter; the diskette's volume number is ignored. Smallest volume number INIT will actually assign to a diskette is 1.

**SEQUENTIAL TEXT FILES**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>As shown</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>.Bx</td>
<td>0</td>
<td>R32767</td>
</tr>
<tr>
<td>Relative Field</td>
<td>.Rx</td>
<td>0</td>
<td>R32767</td>
</tr>
</tbody>
</table>

* With EXEC, always relative to field $0.

**RANDOM-ACCESS TEXT FILES**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>As shown</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record Length</td>
<td>.Lx</td>
<td>1</td>
<td>L32767</td>
</tr>
<tr>
<td>Record Number</td>
<td>.Rx</td>
<td>0</td>
<td>R32767</td>
</tr>
</tbody>
</table>

**BINARY FILES**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>As shown</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting Address</td>
<td>.Ax</td>
<td>A0</td>
<td>A65535</td>
</tr>
<tr>
<td>Number of Bytes</td>
<td>.Lx</td>
<td>1</td>
<td>L32767</td>
</tr>
</tbody>
</table>

**DOS COMMANDS**

<table>
<thead>
<tr>
<th>Command</th>
<th>Quantity</th>
<th>As shown</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR#</td>
<td>slot</td>
<td>PR# s</td>
<td>PR#</td>
<td>PR#</td>
</tr>
<tr>
<td>IN#</td>
<td>slot</td>
<td>IN# s</td>
<td>IN#</td>
<td>IN#</td>
</tr>
<tr>
<td>MAXFILES</td>
<td>files buffers</td>
<td>MAXFILES n</td>
<td>n1 = 1</td>
<td>n1 = 15</td>
</tr>
</tbody>
</table>

Commands use Slot or Drive parameters only when changing to a different Slot or Drive.

If a command omits the Volume parameter or uses V0, the diskette's volume number is ignored. A command that uses the Volume parameter Vv will not be executed unless the diskette's volume number is v.

**HOUSEKEEPING COMMANDS**

- **INIT X [.Vv] [.Sx] [.Dx] [.Vv]**
  Initializes a blank diskette to form a slave diskette. Assigns a unique program name X and volume number Vv (if specified). Saves the BASIC program currently in memory, under file name X.

- **CATALOG [.Sx] [.Dx]**
  Displays volume number and all files on a diskette, with each file's type and sector length. * indicates a locked file.

- **SAVE X [.Sx] [.Dx] [.Vv]**
  Stores current BASIC program onto diskette, under file name X. Overwrites any previous file of same type and name, without warning.

- **LOAD X [.Sx] [.Dx] [.Vv]**
  Loads BASIC program file X into memory, after clearing memory and (if necessary) changing to the correct BASIC.

- **RUN X [.Sx] [.Dx] [.Vv]**
  Loads BASIC program file X, then RUNs the program.

- **RENAME X, Y [.Sx] [.Dx] [.Vv]**
  Changes a diskette file's name from X to Y.

- **DELETE X [.Sx] [.Dx] [.Vv]**
  Erases file X from the diskette.

- **LOCK X [.Sx] [.Dx] [.Vv]**
  Locks file X against accidental change or deletion. LOCKed file shown in CATALOG by X.

- **UNLOCK X [.Sx] [.Dx] [.Vv]**
  Unlocks previously LOCKed file X to allow change or deletion.
VERIY X [.5s] [.D4] [.VV]
Checks file X for internal consistency. If X was saved without a error, no message is given.

MON [.C] [.I] [.D]
Causes display of disk commands (C), input from the disk (I), and output to the disk (O). With no parameters, MON is ignored.

NOMON [.C] [.I] [.D]
Cancels display of disk commands (C), input from the disk (I), and output to the disk (O). With no parameters, NOMON is ignored.

MAXFILES n
Reserves n file buffers for disk input and output (booting reserves 3 file buffers). Use before LOADING or RUNNING a program.

ACCESS COMMANDS

FP [.5s] [.D4] [.VV]
Puts system into Applesoft BASIC, erasing any program in memory.

INT
Puts system into Integer BASIC, erasing any program in memory.

FR# a
Sends subsequent output to slot a. Boots disk if slot a contains disk controller card. FR# sends output to TV screen again.

IN# a
Takes subsequent input from slot a. Boots disk if slot a contains disk controller card. IN# takes input from keyboard again.

CHAIN [.5s] [.D4] [.VV]
RUNs Integer BASIC program file Y, but does not clear variables developed by previous Integer BASIC program.

SEQUENTIAL TEXT FILE COMMANDS

OPEN X [.5s] [.D4] [.VV]
Opens or creates sequential text file X, allocates one file buffer and prepares to WRITE or READ from beginning of file.

CLOSE [X]
Completes WRITE X, if necessary, and deallocates file buffer assigned to text file X. Without file name, CLOSEs all OPEN files (except an EXEC file).

WRITE X [.BB]
Subsequent PRINTs send characters to sequential text file X. WRITEing begins at current file position or (if specified) at byte b. Cancelled by any DOS command.

READ X [.BB]
Subsequent INPUTs and GETs take response characters from sequential text file X, reading begins at current file position or (if specified) at byte b. Input response is one field (all characters to next RETURN). Cancelled by any DOS command.

APPEND X [.5s] [.D4] [.VV]
Opens existing sequential text file X, similar to OPEN, but prepares to WRITE at the end of the file.

POSITION X, Rp
In OPEN sequential text file X, subsequent READ or WRITE will proceed from Rp-th field following current file position.

Executes successive fields in sequential text file X as if typed at keyboard. With Rp parameter, execution begins with p-th field. Fields may include numbered BASIC program lines and direct-execution BASIC or DOS commands to control the Apple.

RANDOM-ACCESS TEXT FILE COMMANDS

OPEN X, Lj [.5s] [.D4] [.VV]
Opens or creates random-access text file X, allocates one file buffer, and defines record length as j bytes. Prepares to WRITE or READ from beginning of Record 0. Same length parameter must be used each time file X is OPENed.

CLOSE [X] [.5s] [.D4] [.VV]
Completes WRITE X, if necessary, and deallocates file buffer assigned to text file X. Without file name, CLOSEs all OPEN files.

WRITE X [.RR] [.BB]
Subsequent PRINTs send characters to random-access text file X. With no parameters, WRITEing begins at current file position. With RR parameter alone, WRITEing starts at byte B of Record R. With Bb parameter, WRITEing starts at byte b of current or specified record. Cancelled by any DOS command.

READ X [.RR] [.BB]
Subsequent INPUTs and GETs take response characters from random-access text file X. With no parameters, READING begins at current file position. With RR parameter alone, READING starts at byte B of Record R. With Bb parameter, READING starts at byte b of current or specified record. Input response is one field (all characters to next RETURN). Cancelled by any DOS command.

MACHINE-LANGUAGE FILE COMMANDS

BSAVE X, Aa, Lj [.5s] [.D4] [.VV]
Stores on diskette, under file name X, the contents of j memory bytes starting at address a.

BLOAD X [.AA] [.5s] [.D4] [.VV]
Loads binary file X into same memory locations from which file was BSAVED or (if specified) starting at address a.

BRUN X [.AA] [.5s] [.D4] [.VV]
BLOADs binary file X, then jumps (Jump) to loaded file's first memory address.