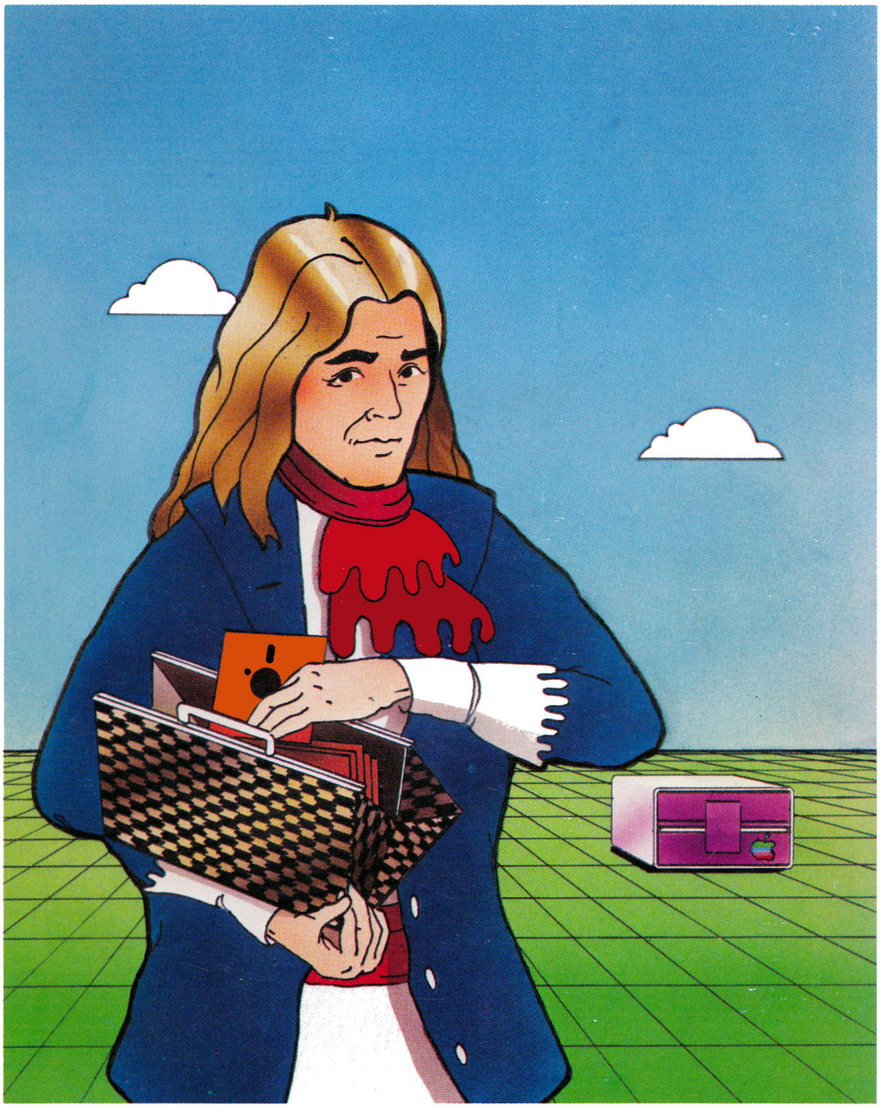
Bag of Tricks



Bag of Tricks™

By Don Worth and Pieter Lechner

QS QUALITY SOFTWARE

Bag of Tricks

By Don Worth and Pieter Lechner

A product of

QUALITY SOFTWARE
6660 Reseda Blvd., Suite 105
Reseda, CA 91335

TABLE OF CONTENTS

Chapter 1 INTRODUCTION

LOADING THE BAG OF TRICKS DISKETTE
THE FOUR PROGRAMS AND WHAT THEY DO
MAKING A PRACTICE DISKETTE

Chapter 2 TRAX - By Pieter Lechner

TRAX TUTORIAL
TRAX COMMANDS
TRAX MESSAGES
SUMMARY OF TRAX COMMANDS

Chapter 3 INIT - By Pieter Lechner

A BRIEF INIT TUTORIAL
INIT OPTIONS
INIT MESSAGES
HOW SECTOR SKEWING CAN AFFECT DISK PERFORMANCE

Chapter 4 ZAP - By Don Worth

A BRIEF ZAP TUTORIAL
ZAP COMMANDS
ZAP ERROR MESSAGES
SUMMARY OF ZAP COMMANDS
ALPHABETICAL LISTING OF ZAP COMMANDS

Chapter 5 FIXCAT - By Don Worth

FIXCAT TUTORIAL
FIXCAT MESSAGES

TABLE OF CONTENTS

Chapter 6 ADVANCED TUTORIALS

IDENTIFYING AND CORRECTING FORMATTING ERRORS
IMPROVING ACCESS TIME FOR DOS DISKETTES
CHANGING VOLUME NUMBERS
FINDING THE A AND L VALUES OF A BINARY FILE
PATCHING DOS USING ZAP
EXAMINING/CHANGING BASIC'S RANDOM TEXT FILES
RELEASING UNUSED SPACE IN FILES
SCANNING A DISK FOR I/O ERRORS
LOCATING AND FIXING AN I/O ERROR
COPYING PASCAL FILES TO DOS USING ZAP MACROS
COMPARING FILES USING ZAP MACROS
PATCHING A MACHINE LANGUAGE PROGRAM USING ZAP
RECOVERING LOST SECTORS IN THE VTOC FREESPACE MAP
RECLAIMING TRACKS 1 AND 2 FOR FILESPACE
A DOS-LESS BOOT PROGRAM
UN-DELETING A FILE
RECONSTRUCTING A BLOWN CATALOG

Appendix A DISCUSSION OF DISK I/O ERRORS

ACKNOWLEDGEMENTS

The concept for ZAP originated with a similar program written for the IBM 360 by Vic Tolomei and myself almost ten years ago. Thanks to Pete Nielsen, Lou Rivas, and Dave Robertson for their suggestions, criticisms and product testing.

Don D. Worth

My thanks to Greg Staie and Dave Garson for their assistance, and a special thanks to Will Greaves, who developed the original idea for TRAX.

Pieter M. Lechner

CHAPTER 1 INTRODUCTION

Bag of Tricks is a collection of utility programs inspired by some simpler programs provided in an appendix of our book, Beneath Apple DOS. After the book was written we received many requests for more complete programs for use in conjunction with the information presented in Beneath Apple DOS. In answer to these requests, we have developed four machine language utility programs which encompass most of the functions needed by Apple II disk owners for diskette manipulation as well as error detection and recovery. Each of these programs includes features that are not available in other disk utilities that are currently available for the Apple II.

With Bag of Tricks our intent was to create powerful and easy to use utilities of value to both the very technical user and the average user. Our approach was to write programs we ourselves wanted to use. During development we started by adding all the features we needed and added all the features we imagined anyone else might want. This approach has generally led to more functional extensions than one might normally find in such programs. It is hoped that beginners will not find this intimidating. Every effort has been made to document the programs carefully, providing many examples of their use, so that expert and novice alike can profit from them.

LOADING THE BAG OF TRICKS DISKETTE

Bag of Tricks consists of four separate programs provided on a single diskette. They each require a 48K APPLE II or APPLE II PLUS with at least one disk drive. A printer interface (in any slot) is optional. Both 13 sector diskettes (DOS 3, DOS 3.1, DOS 3.2) and 16 sector diskettes (DOS 3.3, CP/M, and PASCAL) are supported by most of the programs.

To load the Bag of Tricks diskette, merely insert it in drive 1 and power the machine on (AUTOSTART ROM) or type PR#6 (if the disk card is plugged into slot 6) from BASIC. After a moment you will see a menu listing the four programs. Press the key that corresponds to the first letter of the program that you wish to use. A "LOADING" message will appear for a short time, then the program you selected will become available.

CAUTION: The Bag of Tricks diskette cannot be copied and attempting to write to the diskette will destroy the programs. Bag of Tricks is provided on a high quality diskette that should give you trouble free usage for many years. If the diskette should fail, it will be replaced promptly by Quality Software. Simply return it to Quality Software, 6660 Reseda Blvd., Suite 105, Reseda, CA 91335. If you have owned the diskette for less than 90 days, include your receipt and there will be no charge for replacement (only those outside the continental United States need pay shipping). If you have owned the diskette for more than 90 days, there is a nominal charge of $5.00 plus shipping. Shipping charges for replacing a diskette are the same as those for a back up copy and may be found at the end of this chapter. California residents must add 6% sales tax.

Bag of Tricks owners who feel they need an immediate back up copy may purchase one for $5.00 plus shipping. Simply fill out and send in the coupon at the end of this chapter along with payment directly to Quality Software. Only one back up copy may be ordered by each owner, and phone orders are not accepted.

THE FOUR PROGRAMS AND WHAT THEY DO

Although the four programs that comprise Bag of Tricks can often be used together to perform some task, the purpose of each is quite different and the prospective user need not feel that he must have a use for all four programs, or all of the functions of any one. The programs provided are briefly described as follows:

TRAX:
A track examination program. TRAX will read a track from a diskette in its "raw" pre-nibbilized form and format it on the screen, attempting to pick out the sector formatting. If the diskette is non-standard in its formatting, such as a protected diskette or one which has been damaged in some way, TRAX will highlight its anomolies. TRAX is also useful in conjunction with the INIT program to determine the physical order or skewing of sectors on a diskette.

ARMED WITH YOUR BAG OF TRICKS, YOU NEED NOT FEAR "SNIDELY DISKCRASH"

INIT:
The INIT program can be used to reformat a single track on a diskette, a range of tracks, or the entire diskette. In addition, INIT will optionally attempt to preserve the contents of any readable sectors it finds before reformatting. Thus, INIT can be used to fix a single sector whose formatting has been damaged so that it can no longer be read from or written to. This avoids having to initialize the entire diskette with the DOS INIT command. INIT will also allow you to specify the order of the sectors on any given track. Doing this can improve disk read times by about 40%.

ZAP:
ZAP in its simplest sense allows you to read and modify a diskette at the track and sector level. A sector may be read and displayed in hexadecimal and ASCII and, optionally, modified and rewritten to the disk. ZAP provides over 50 commands, including some programmability with macros, labels, and loops, allowing you to perform complex manipulations on diskettes. Full support exists for DOS files of all types, CP/M files, and PASCAL files as well. ZAP is perhaps the most complex of the four programs but you will probably find you use it most heavily.

FIXCAT:
The FIXCAT program is an automated utility which allows you to diagnose and correct errors in the catalog track of any DOS diskette. In addition, it takes the Find Track/Sector lists program (FTS) of Beneath Apple DOS a step further by actually recovering lost files on a diskette **automatically**! FIXCAT will also allow you to remove the DOS image from tracks 1 and 2 to provide more room for files and will recover lost sectors by correcting the VTOC freespace map.

It is the authors' belief that learning is best accomplished by example. Therefore we have provided example sessions for each of the four programs that comprise Bag of Tricks. In the next four chapters each of the programs is covered separately, including both a detailed description and a tutorial. The remainder of this book (Chapter 6) is devoted to useful examples, some of which involve more than one of the individual programs. The techniques provided in Chapter 6 not only demonstrate Bag of Tricks but also should be of value in fixing diskettes or patching DOS. We suggest that you first read over the descriptions of each program and work through the associated tutorials on your own Apple before attempting the procedures covered in Chapter 6. However, a complete understanding of each of the programs is not neccessary to successfully use Chapter 6.

MAKING A PRACTICE DISKETTE

Most of the tutorials given are based on a "practice diskette". Although any diskette can be used, it is recommended that you construct a diskette using the following procedure. This way you will see the same displays that are described in the tutorials and you won't risk damage to one of your important diskettes.

To construct the practice diskette, put the Bag of Tricks diskette aside and follow these simple steps:

STEP 1 - Boot DOS (DOS 3.3 if you have it) using your Master Diskette
STEP 2 - Select APPLESOFT with the FP command
STEP 3 - Enter the following HELLO program:

```
NEW
10 END
```

STEP 4 - INIT a blank diskette as follows:

```
INIT HELLO,D1,V1
```

STEP 5 - BSAVE some memory to create a binary file:

```
BSAVE BINARY FILE,A$3D0,L$30
```

STEP 6 - Enter, save and run the following program:

```
NEW
10 D$=""   (insert a CTRL D between the quotes)
20 PRINT:PRINT D$;"OPEN TEXT FILE"
30 PRINT D$;"WRITE TEXT FILE"
40 PRINT "1,2,3"
50 PRINT "4,5,6"
60 PRINT "7,8,9"
70 PRINT D$;"CLOSE TEXT FILE"
80 END
SAVE APPLESOFT PROGRAM
RUN
```

Set this practice diskette aside. As you work through the tutorials, use this diskette whenever the text calls for the "practice diskette".

TO GET A BACK UP COPY OF BAG OF TRICKS

Each Bag of Tricks owner has the option of purchasing **one** back up diskette. If you wish to have a duplicate copy for immediate back up, you must mail the coupon on the next page directly to Quality Software along with payment of $5.00 and shipping charges. The back up diskette is identical to the original but does not include documentation.

SHIPPING CHARGES:
48 Continental United States (UPS).......$ 1.50
Alaska, Hawaii (air mail)................$ 5.00
Canada and Mexico (insured air mail).....$ 6.00
Other countries (insured air mail).......$ 8.00

PAYMENT:
Make check or bank draft payable to Quality Software.
VISA or MASTERCARD users please specify expiration date.
No CODs. California residents must add 6% sales tax.

IMPORTANT:
When you send in the coupon below, please first tear this page out of the book and do not remove the torn, perforated edge from the side of the coupon. If you send a copy of the coupon or remove the perforated edge from the coupon, we will be unable to process your order.

....

Please send me a back up copy of Bag of Tricks™

Name...

Address..

City,State,Postal Code.......................................

Country..

Check No. .... or

Bankcard #.........................expires...............

mail to: QS QUALITY SOFTWARE
6660 Reseda Blvd., Suite 105, Reseda, CA 91335
(213) 344-6599

CHAPTER 2
TRAX — By Pieter Lechner

We have put TRAX first in the order of the Bag of Tricks programs because it deals with the diskette at its most primitive level. The fact is, however, that TRAX is the most technically challenging of the four Bag of Tricks programs. Although it is not hard to use TRAX, its applications are harder to appreciate, and we therefore recommend that beginners become familiar with the other Bag of Tricks programs, especially ZAP, before trying to use TRAX.

The purposes of TRAX are twofold. First, it provides you with the ability to locate errors on a diskette rapidly and to determine the extent of the damage. Second, TRAX provides you with the means of examining how data actually appears on a diskette, which is useful in better understanding DOS. While TRAX is designed to operate on normal 13 or 16 sector diskettes, its data reading technique allows it to read any diskette created on an Apple disk drive.

Because some of you will no doubt use TRAX to look at a "protected" diskette, a few things should be made clear. Dumping an entire track of raw, unnibblized data into memory is relatively simple (a short program to perform that task was included in our book Beneath Apple DOS). Interpreting the raw data is a much more complicated task. Interpretation becomes even more difficult when very few assumptions can be made about the nature of the data. This fact, along with our desire not to produce a means of breaking disk protection schemes, led to the following decision. TRAX assumes that the diskette being examined is a normally formatted diskette that may have sustained some damage during usage. It is possible to examine protected diskettes using TRAX, but we are neither advocating nor supporting such usage.

Throughout this chapter we will use such terms as raw data and nibblizing. If these terms are not at all familiar to you, we recommend that you read Chapter 3 of our book, Beneath Apple DOS, which is a fairly complete discussion of Apple II diskette formatting.

A brief explanation of how TRAX works is in order. Raw data is read from the desired track one byte at a time and stored in memory. Enough bytes are read to guarantee reading the track at least three times. This allows TRAX, in most cases, to find at least one good track image. The format of a track on a normal Apple diskette is reasonably well defined, and is described in Figure 2.1.

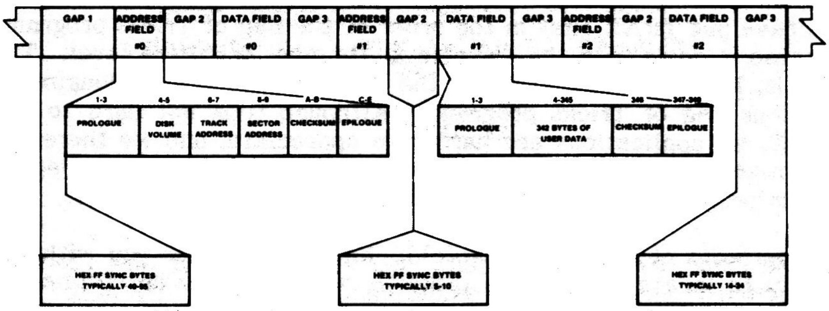


FIGURE 2.1 — TRACK FORMAT

TRAX attempts to identify the different segments of the track by locating each gap (gap 1, gap 2, and gap 3 of Figure 2.1). This technique, if successful, allows TRAX to recognize any abnormality in either an address field or a data field. When all segments of the track have been located, each is analyzed. The various address fields are decoded and any deviation from the norm is indicated on the screen. In its raw form, the address field appears like Figure 2.2.

PROLOGUE	VOLUME	TRACK	SECTOR	CHECKSUM	EPILOGUE
D5 AA 96	XX YY	XX YY	XX YY	XX YY	DE AA EB

ODD-EVEN ENCODED

DATA BYTE — $D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0$
XX — $1\ D_7\ 1\ D_5\ 1\ D_3\ 1\ D_1$
YY — $1\ D_6\ 1\ D_4\ 1\ D_2\ 1\ D_0$

FIGURE 2.2 — ADDRESS FIELD

Each of the data fields is also decoded, and any deviations from the norm are indicated on the screen. In its raw form, the data field appears like Figure 2.3.

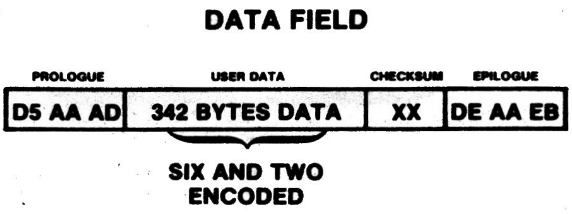


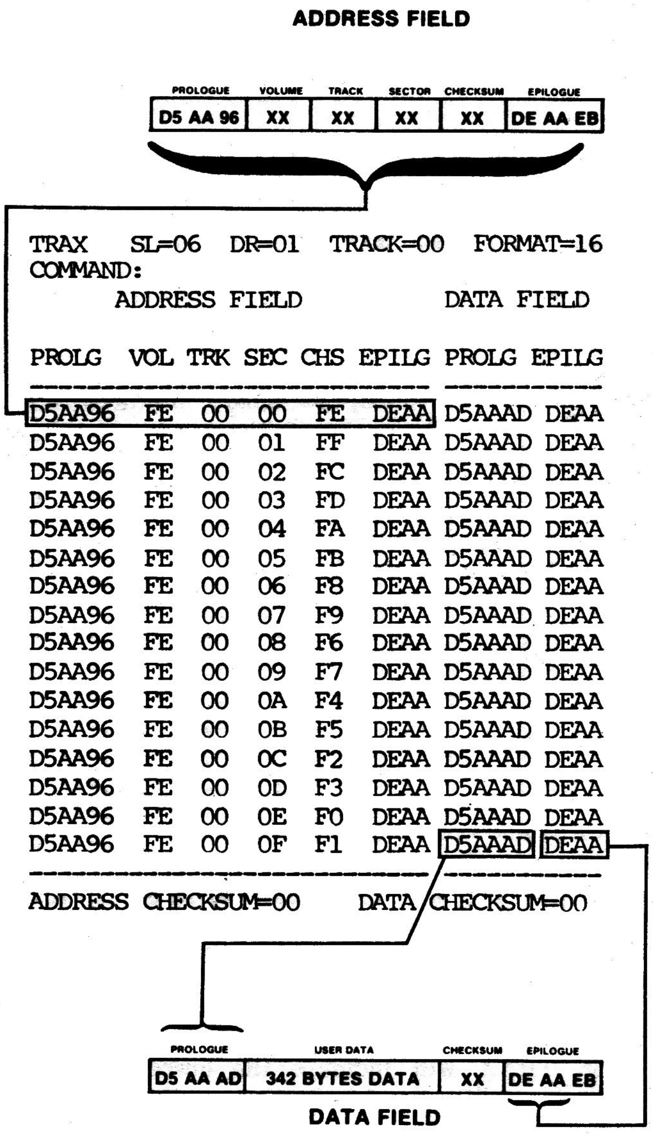
FIGURE 2.3 — DATA FIELD

If the gaps cannot be found, secondary methods are used to determine if any valid data exists on the track. If this also fails, TRAX will display the message "UNABLE TO INTERPRET DATA" on the screen. Some of the reasons that TRAX might fail to interpret data are that the track has never been formatted, or the track is so badly damaged that no recognizable portions remain. While TRAX will not correctly diagnose every type of I/O error, it will uncover formatting errors a high percentage of the time. In addition, TRAX should prove to be informative to beginners and a useful tool for experienced users.

A TRAX TUTORIAL

The following tutorial is provided to familiarize the first-time user with the features and operation of TRAX.

After booting the Bag of Tricks diskette, select TRAX from the menu. Then remove the Bag of Tricks diskette and insert the practice diskette in your disk drive. Look at the top line of the TRAX display screen. Included there is the track number, which is currently 00. To read the indicated track number into memory, press R on your keyboard. TRAX will begin reading track 0. While it is working, TRAX will indicate the current operation being performed, first reading the track, then analyzing the data, and finally displaying the results. Your screen should be similar to the center of Figure 2.4



```
TRAX    SL=06  DR=01   TRACK=00   FORMAT=16
COMMAND:
      ADDRESS FIELD              DATA FIELD

PROLG   VOL  TRK  SEC  CHS  EPILG  PROLG  EPILG
-------------------------------------- ----------
D5AA96   FE   00   00   FE   DEAA  D5AAAD  DEAA
D5AA96   FE   00   01   FF   DEAA  D5AAAD  DEAA
D5AA96   FE   00   02   FC   DEAA  D5AAAD  DEAA
D5AA96   FE   00   03   FD   DEAA  D5AAAD  DEAA
D5AA96   FE   00   04   FA   DEAA  D5AAAD  DEAA
D5AA96   FE   00   05   FB   DEAA  D5AAAD  DEAA
D5AA96   FE   00   06   F8   DEAA  D5AAAD  DEAA
D5AA96   FE   00   07   F9   DEAA  D5AAAD  DEAA
D5AA96   FE   00   08   F6   DEAA  D5AAAD  DEAA
D5AA96   FE   00   09   F7   DEAA  D5AAAD  DEAA
D5AA96   FE   00   0A   F4   DEAA  D5AAAD  DEAA
D5AA96   FE   00   0B   F5   DEAA  D5AAAD  DEAA
D5AA96   FE   00   0C   F2   DEAA  D5AAAD  DEAA
D5AA96   FE   00   0D   F3   DEAA  D5AAAD  DEAA
D5AA96   FE   00   0E   F0   DEAA  D5AAAD  DEAA
D5AA96   FE   00   0F   F1   DEAA  D5AAAD  DEAA
-------------------------------------- ----------
ADDRESS CHECKSUM=00        DATA CHECKSUM=00
```

FIGURE 2.4 — ADDRESS AND DATA FIELD DISPLAY

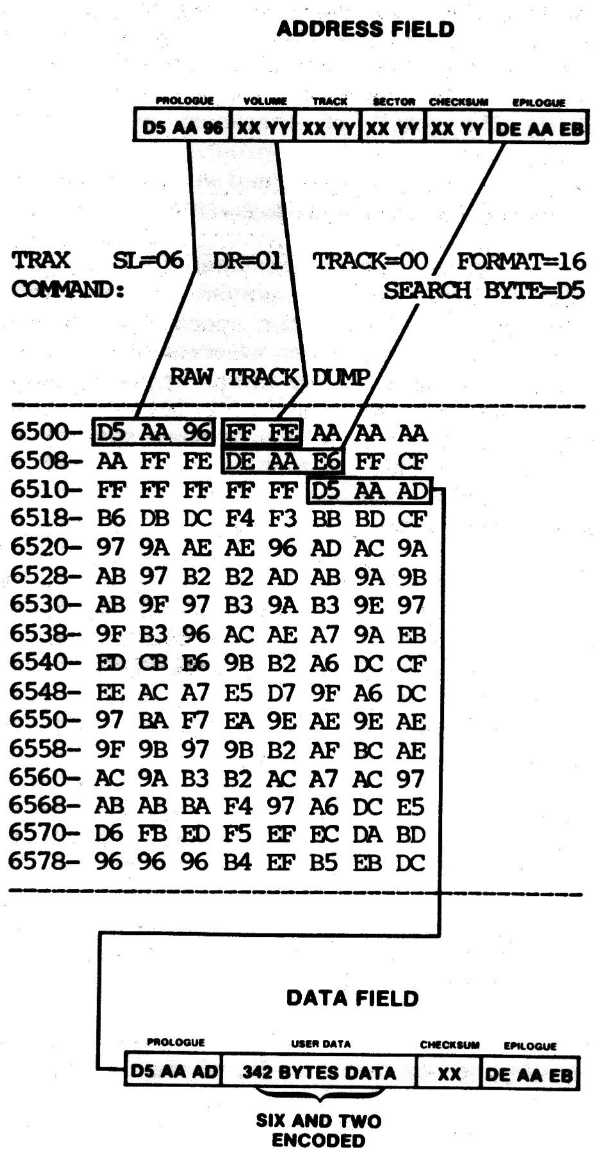
As the figure indicates, most of the vital information used by DOS to locate and read data is displayed for you by TRAX. It is important that this information is correct, because if a single byte is altered the entire sector is unreadable by DOS. The checksums displayed at the bottom of the screen are not read from the diskette but are actually the result of checksum computations. These computations are performed on the appropriate data in a way similar to the way that DOS performs them, and must provide a zero result (00) for valid address and data fields.

Because you are using a normal, presumably undamaged diskette, there should be no abnormalities displayed on the screen. If there were you would hear a bell and the appropriate screen location would be displayed in inverse. When an error occurs in a checksum computation, and that error is the same for every sector on the track (unlikely in normal usage), the erroneous value is displayed in inverse. Should an error occur in one or more sectors, but not the same error on every sector, a pair of inverse asterisks (**) is displayed. In this case, you may access the actual checksum computations with additional keypresses.

Now press the A key. The column in the middle of the display, the one labeled CHS, will change to inverse and will display the address field checksum computations for each sector. Each of these checksums, which are computed by exclusive-ORing the four data values in each address field, must be zero for a normal, undamaged diskette. The screen can be returned to the normal display by pressing the RETURN key.

Now press the D key. The column in the middle of the display will again change to inverse, but this time it will display the results of the data field checksum computations for each sector. Again, each of these checksums must be zero for a normal, undamaged diskette. Pressing RETURN will restore the screen to normal. For a detailed explanation of the construction of address and data fields, see Beneath Apple Dos (Chapter 3).

After restoring the screen to its initial display, press the X key. This places you in the hex dump mode, where the raw dump of the track is displayed. Figure 2.5 is a typical raw dump display, and points out how address fields and data fields can be identified in the raw data.



```
TRAX    SL=06  DR=01  TRACK=00  FORMAT=16
COMMAND:                  SEARCH BYTE=D5

           RAW TRACK DUMP
------------------------------------------
6500- D5 AA 96 FF FE AA AA AA
6508- AA FF FE DE AA E6 FF CF
6510- FF FF FF FF FF D5 AA AD
6518- B6 DB DC F4 F3 BB BD CF
6520- 97 9A AE AE 96 AD AC 9A
6528- AB 97 B2 B2 AD AB 9A 9B
6530- AB 9F 97 B3 9A B3 9E 97
6538- 9F B3 96 AC AE A7 9A EB
6540- ED CB E6 9B B2 A6 DC CF
6548- EE AC A7 E5 D7 9F A6 DC
6550- 97 BA F7 EA 9E AE 9E AE
6558- 9F 9B 97 9B B2 AF BC AE
6560- AC 9A B3 B2 AC A7 AC 97
6568- AB AB BA F4 97 A6 DC E5
6570- D6 FB ED F5 EF EC DA BD
6578- 96 96 96 B4 EF B5 EB DC
------------------------------------------
```

FIGURE 2.5 — RAW TRACK DUMP

While the raw track image is difficult to decode manually, it is nevertheless informative and useful in viewing tracks that TRAX was unable to interpret. TRAX stores the track image in a buffer in memory starting at $6500 and ending at $7FFF. The end of the buffer is padded with FF's to insure consistency for different size track images.

TRAX allows you to move freely through the track image. The arrow keys will scroll one line at a time. The left arrow scrolls one line backward in the buffer, and the right arrow scrolls one line forward. It is impossible to scroll past either the start or end of the buffer, and any attempt to do so will be disregarded. To scroll forward one full screen (80 Hex Bytes), press the N key (for Next). To scroll backward one full screen, press the P key (for Previous). You may also move to the beginning of the buffer by pressing B and to the end of the buffer by pressing E. Experiment briefly to get a feel for how to move about within the buffer. When you are done, press B to go to the beginning of the buffer.

TRAX also allows you to search for a particular byte. This is done with the L key. Notice the top of the display identifies the current search byte as D5. This is the default value, because D5 is the byte which normally begins all address and data fields. Each time you press the L key the next D5 in the track image will move to the top line on the screen. In this manner you can very rapidly locate and examine all address and data fields in the buffer. If you wish, you may change the search byte by pressing C. A cursor will appear appear on the value of the current search byte (the D5 at the end of the command line), allowing you to enter any valid byte (80-FF Hex). Experiment with.other values for search byte, then exit the raw dump mode by pressing the X key. You will be returned to the address and data field display (the analysis mode).

When you are using TRAX to analyze diskettes, you should keep in mind that the sector order in the address and data field display is the same as the order of the data in the raw data buffer. This fact will, in many cases, help you determine what portion of a damaged diskette has been damaged. Thus, it is not uncommon in an analysis session to find yourself switching in and out of the raw data mode.

Another point to keep in mind is that the sector numbers on the TRAX display are the actual numbers on the diskette, which we refer to in Chapter 2 as the physical sector numbers. The logical sector numbers, which are referred to by the DOS file manager, are always different in DOS 3.3.

Just as you can move around within a raw track buffer when in the raw dump mode, you can also move across the diskette when in the analysis mode, choosing any track that you wish to examine. Press the N key to access the Next track. This is like pressing R except that the track indicator is incremented before the read is performed. When the read is complete, you will see that the track number is now 1.

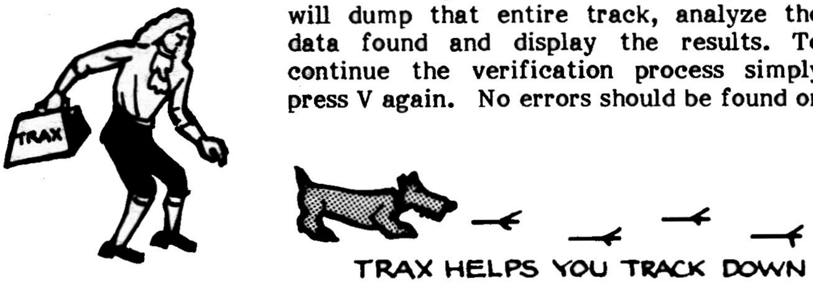
To access the Previous track, press the P key. If you do this you will find that you are back to track 0.

To select a particular track, the arrow keys are used. For example, select track 11 (Hex) by repeatedly pressing the right arrow key. The track number at the top of the screen will increase by one with each keypress. Stop when it reaches the value 11. You should have noticed that when you first changed the track number, it was no longer displayed in inverse. This is to indicate that the current display does not correspond with the track indicator. If you change the track indicator back to what it was originally, it would again be displayed in inverse.

Having selected track 11, press the R key to read, analyze, and display the results. The display should be similar to the earlier tracks you looked at, with only the track number and checksum bytes changing at all. In fact on normal undamaged diskettes there should be little difference between individual tracks.

As you can see, it is simple to check a particular track that you suspect of having an I/O Error. Often, however, you may not know where the error is located, only that it exists. TRAX can very quickly locate abnormalities on a diskette. The following exercise shows how you verify that a diskette is free of formatting errors.

Press the 0 on your keyboard, this will recalibrate the disk arm and read track 0. Now simply press V to verify the diskette. TRAX will verify that all sectors are readable, starting with the next track. If any sector can't be read, TRAX will dump that entire track, analyze the data found and display the results. To continue the verification process simply press V again. No errors should be found on



the diskette you are using for this tutorial, and you should see a message indicating that the verification process has been completed.

This ends the TRAX tutorial. A thorough discussion of each of the TRAX commands follows.

TRAX COMMANDS

There are two sets of TRAX commands. The first set is available when the address and data field display is on the screen. We refer to this as the **analysis mode.** The second set is available when the raw track dump is on the screen. We refer to this as the **raw dump mode.**

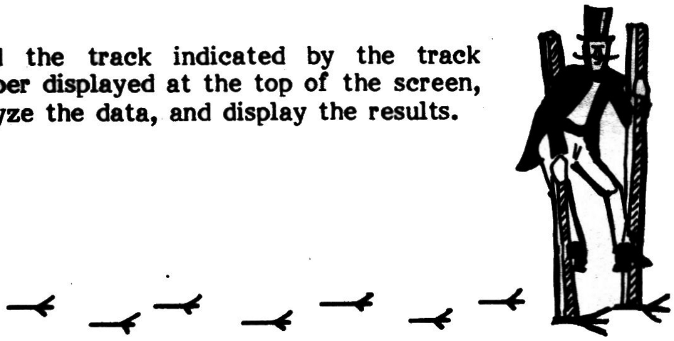
ANALYSIS MODE COMMANDS

ARROW KEYS

In the analysis mode, the arrow keys are used to select the track number to read. The right arrow key will increment the current track indicator and the left arrow will decrement the track number. If the track number is displayed in inverse, this indicates that the data on the screen refers to that track. If the track number is not displayed in inverse, the screen data does not correspond to the indicated track number.

R

Read the track indicated by the track number displayed at the top of the screen, analyze the data, and display the results.


THE SOURCE OF DISK ERRORS

N

Read the next track (i.e. the current track indicated plus one), analyze the data, and display the results. The wraparound feature causes track 00 (Hex 00) to be read after track 34 (22 Hex).

P

Read the previous track (i.e. the current track indicated less one), analyze the data, and display the results. The wraparound feature causes track 34 (22 Hex) to be read after track 00 (Hex 00).

F

Toggle between 13 and 16 sector formats. While TRAX will in many cases identify the format of a particular diskette, the user must change the format setting manually.

V

Verify the diskette starting at the next track and continuing through track 34 (Hex 22). Any abnormalities are displayed, stopping the verification process. The verification process may be continued by pressing V again.

A

Display the result of the checksum computations for the address fields of each sector. The results are shown in the center column of the display and are normally zero. The computation consists of exclusive ORing the four values in the address field. After this command has been used, pressing RETURN will return the screen to its normal display.

D

Display the result of the checksum computations for the **data** fields of each sector. The results are shown in the middle column of the display and are normally zero. The computation, described in detail in our book Beneath Apple DOS, essentially involves exclusive ORing the values in the data field. After this command has been used, pressing RETURN will return the screen to its normal display.

0

Force recalibration of the disk arm followed by a read, analysis, and display of track 0.

S

Allows the selection of a different SLOT and DRIVE configuration. After pressing S, the word SLOT will flash on the command line. The arrow keys may be used to select any slot that contains a Disk II compatible controller card. Press RETURN when the SLOT selection is made. The word DRIVE will then appear on the command line, and you may select either drive 1 or drive 2 using the arrow keys. Press RETURN to complete the selection process.

X

Exit the analysis mode and enter the raw dump mode, which displays the raw hex dump of the current track.

ESCAPE

Exit the TRAX program and return to the Bag of Tricks main menu. The Bag of Tricks diskette should be in drive 1 of the selected slot. An option to return to the program is offered to prevent the user from accidentally exiting.

RIGHT ARROW

Scroll forward one line (eight hex bytes) in the hex display of the track.

LEFT ARROW

Scroll backward one line (eight hex bytes) in the hex display of the track.

N

Scroll forward one page (80 hex bytes) in the hex display of the track.

P

Scroll backward one page (80 hex bytes) in the hex display of the track.

B

Go to the begining of the track dump buffer.

E

Go to the end of the track dump buffer.

C

Change the current search byte, the value for which is displayed on the right side of the command line. Enter any valid byte (80-FF Hex) and press the RETURN key. Pressing RETURN after typing C leaves the value of the search byte unchanged.

L

Starting with the second line on the screen, look for the current search byte (default D5) displayed on the right side of the command line. If found, the desired byte will be displayed on the top line of the raw hex dump. If the search is unsuccessful, an error message appears.

X

Exit the raw dump mode and return to the analysis mode, which redisplays the address and data field information.

TRAX MESSAGES

Although TRAX is accurate in most cases, some unforeseen types of I/O errors may occur that will not be correctly interpreted. TRAX is meant to be a tool in aiding the user to determine the nature and location of damaged areas of a diskette, but we do not guarantee that TRAX can uncover every possible type of diskette error.

READING DATA

This message is displayed when the disk drive is working. The drive will be turned on and a brief delay will occur to allow the motor to come up to speed. Then the track is dumped into a large buffer area one byte at a time.

ANALYZING DATA

A track has been read in and now TRAX is attempting to determine if any part of the track has been damaged. When the analysis is complete, the results are displayed on the address and data field display.

VERIFYING DISKETTE

This indicates that tracks are being verified to insure that all sectors on those tracks are readable. If an abnormality is found, TRAX will stop the verification process and display the analysis of the track in question.

VERIFICATION COMPLETE

This indicates that you are using the V command and you have reached the end of the diskette. No abnormalities were found on the diskette between the track where the V command was entered and the end of the diskette.

APPEARS TO BE 13 SECTOR FORMAT

This message will probably occur if Format is set to 16 and a 13 sector disk is used. In the case of certain damaged diskettes, TRAX might not be able to distinguish between 13 and 16 sector formats.

APPEARS TO BE 16 SECTOR FORMAT

This message will probably occur if Format is set to 13 and a 16 sector disk is used. In the case of certain damaged diskettes, TRAX might not be able to distinguish between 13 and 16 sector formats.

UNABLE TO INTERPRET DATA

TRAX has been unable to find any valid data on the track. In the case of a normal diskette, this is a strong indication that no recoverable data remains. In the case of a "protected" diskette this simply indicates that the format being used is different enough from a normal diskette that TRAX does not recognize it.

UPDATED

This prompt will be displayed to indicate that a 13 sector diskette has been updated to contain a sector written in 16 sector format. This is commonly done to allow a diskette to boot on a disk controller card equipped with either a 13 sector or 16 sector prom.

NO DATA

This will generally only be displayed on 13 sector diskettes indicating a sector that has no data field. (The DOS initialization process for 13 sector diskettes never writes the data field, whereas a zeroed data field is written for 16 sector diskettes.) It is possible for a data field to be damaged in such a way that the data field cannot be found, in which case it is probably not recoverable.

DAMAGED

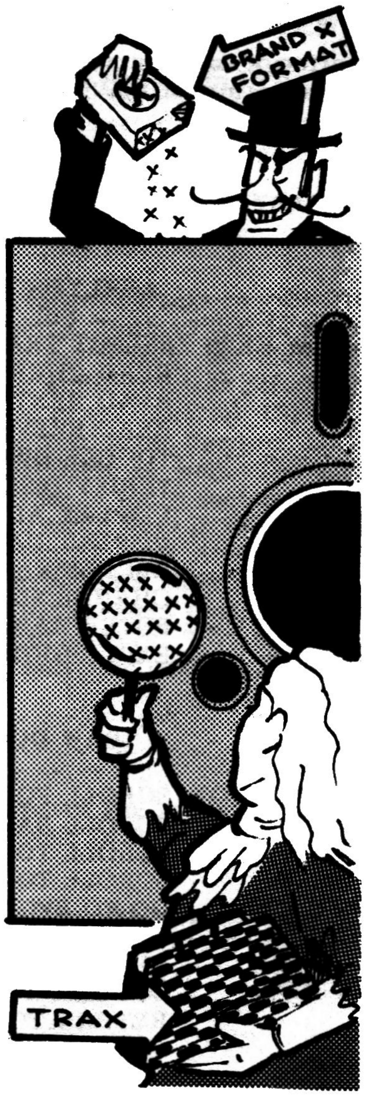
This indicates that TRAX could not find enough of an address or data field (in some cases both) to make a meaningful display. This generally indicates that the particular data is not recoverable.

BYTE NOT FOUND

A search, using the L command in the raw dump mode, has failed. The search byte was not found between the current buffer location and the end of the buffer.

TOP OF BUFFER

An attempt has been made to move backward in the raw dump buffer and the top of the buffer has been reached.



TRAX ALLOWS YOU TO INSPECT DISKETTES FOR NON-STANDARD FORMATTING

END OF BUFFER

An attempt has been made to move forward in the raw dump buffer and the bottom of the buffer has been reached.

SUMMARY OF TRAX COMMANDS

ANALYSIS MODE COMMANDS

ARROW KEYS	SELECT TRACK TO READ
R	READ CURRENT TRACK
N	READ NEXT TRACK
P	READ PREVIOUS TRACK
F	TOGGLE DISK FORMAT (13 OR 16 SECS)
V	VERIFY DISK AND DISPLAY ERRORS
A	DISPLAY ADDRESS CHECKSUMS
D	DISPLAY DATA CHECKSUMS
0	RECALIBRATE AND READ TRACK 0
S	CHANGE SLOT/DRIVE CONFIGURATION
X	ENTER RAW DUMP MODE
ESC key	EXIT PROGRAM

RAW DUMP COMMANDS

RIGHT ARROW	SCROLL FORWARD ONE LINE
LEFT ARROW	SCROLL BACK ONE LINE
N	SCROLL TO NEXT PAGE
P	SCROLL TO PREVIOUS PAGE
B	GO TO BEGINNING OF BUFFER
E	GO TO END OF BUFFER
C	CHANGE SEARCH BYTE
L	LOOK FOR SEARCH BYTE
X	RETURN TO ANALYSIS MODE

CHAPTER 3
INIT — By Pieter Lechner

The INIT utility allows you to format a range of tracks on a diskette — from a single track to an entire diskette. Both 13 and 16 sector formats are supported. Additional options include the ability to preserve any readable sectors on a track, allowing you to reformat a damaged track without affecting the rest of the diskette. INIT was, in fact, originally developed to perform this latter function, but it can provide a much broader range of uses as well. The most important is its ability to choose the order in which sectors appear on the track. This subject is covered in detail later in this chapter, under the heading HOW SECTOR SKEWING CAN AFFECT DISK PERFORMANCE. By selecting the appropriate skewing you can greatly improve the speed with which DOS accesses files, especially when reading programs from diskette into memory. INIT allows you to update your diskettes, changing only the skewing, thereby speeding most reads from the diskette.

A BRIEF INIT TUTORIAL

The following tutorial will serve to introduce you to INIT's features. Boot up the Bag of Tricks diskette, invoke INIT, and follow along with the tutorial.

Upon entry to INIT you will see a screen that looks like Figure 3.1.

```
                    * * I N I T * *

          DISK SECTORING  >16
             DISK FORMAT   DOS
           PRESERVE DATA   YES
          SKEW DIRECTION   DESCENDING
             SKEW FACTOR   02
                    SLOT   06
                   DRIVE   01
           VOLUME NUMBER   DEFAULT
          STARTING TRACK   00-DEC  00-HEX
            ENDING TRACK   34-DEC  22-HEX



             * SPACE BAR MOVES CURSOR *
            * ARROW KEYS CHANGE VALUES *
          * RETURN CONCLUDES DATA ENTRY *
              * ESCAPE EXITS PROGRAM *
```

FIGURE 3.1 — ORIGINAL INIT SCREEN

During this tutorial you will use INIT to format an entire diskette, so at this point remove the Bag of Tricks diskette and place a blank diskette in your disk drive. If you have two disk drives you may use either drive. You will notice a small arrow (>) following the words DISK SECTORING and just in front of the value 16. This is the cursor, indicating the line on which you may enter data. Press the space bar and you will see that the cursor has moved to the second line, containing the words DISK FORMAT. The space bar is the means by which you may position the cursor to any input parameter line on the screen. If you continue pressing the space bar you will be able to move the cursor until it is back in its original position. Please do that now. Your screen should now be identical to Figure 3.1 above. If it is not, please continue pressing the space bar until it is.

Since data to be input to INIT is well defined, it was possible to eliminate the need for you to type out any of the input parameters. Instead, all possible options may be selected by simply using the right and left arrow keys (—> and <—). If you now press the right arrow key, you will see the DISK SECTORING value, 16, change to 13. Since there are only two possible inputs for this line, either arrow key will change the value displayed. Please select the appropriate value for your system. This tutorial will assume that you are using DOS 3.3 (16 sector format) although any differences for DOS 3.2 (13 sector format) will be noted. When you have selected the number of sectors you wish to use, press the space bar to move to the next line. The cursor should now be in front of the DISK FORMAT default, DOS, which is the value we will use in this example. You may wish to see what other options are available by using the arrow keys at this time. When you are ready to continue, make sure the word DOS appears and then press the space bar to move on to the next line.

You are now on the line labeled PRESERVE DATA and the cursor should be in front of the default, YES. Since we are formatting a blank diskette, there is no need to try to preserve any data, so, using either arrow key, select the value NO. (It should be noted that it is possible to format a blank diskette by answering YES to this question but it would take considerably longer than necessary since the program must search for data on every track. If any data is found it will be preserved, thereby producing a copy rather than a freshly formatted diskette.) Making sure that you have selected the value NO, you may move to the next line using the space bar.

The cursor should now be in front of the word DESCENDING on the line reading SKEW DIRECTION. This is the value we want to use for this example so we can move on to the next line. Press the space bar once to move to the next line. The cursor is now in front of the value 2. This is the skew factor for a normal DOS 3.3 diskette. On the other hand, a skew factor of 9 is the optimal skew factor for most disk LOAD or BLOAD operations, so we will use 9. The right arrow key increments the value and the left arrow key decrements it. A wrap around feature is built in so that a value may be selected using either arrow key. Press either arrow key until the value 9 appears. Then press the space bar to move to the next line.

You will now select the SLOT number of your disk controller card, usually slot 6. When you have made your selection, press the space bar to move on to DRIVE information. Select the drive number (1 or 2) and continue by pressing the space bar again. You will now select VOLUME number. Leaving the current value of DEFAULT will result in the volume number 254 ($FE in hexadecimal). This is the value DOS uses if you INIT a diskette without specifying a volume number. You may select any volume number you wish, then move on to the next line.

The cursor should be at the line labeled STARTING TRACK. The value of 00 now appearing there is correct, so you may continue by using the space bar again. The last line for data input, ENDING TRACK, also has the correct value (34), so you are now done entering data. You may press the RETURN key indicating that all data entry is finished. You will see the cursor on the line asking the question "IS THE ABOVE OK ?" followed by the answer YES. Check carefully to make sure all the questions have been answered correctly. At this point your screen might look like this:

```
              * * I N I T * *

        DISK SECTORING   16
           DISK FORMAT   DOS
         PRESERVE DATA   NO
        SKEW DIRECTION   DESCENDING
           SKEW FACTOR   09
                  SLOT   06
                 DRIVE   01
         VOLUME NUMBER   DEFAULT
        STARTING TRACK   00-DEC   00-HEX
          ENDING TRACK   34-DEC   22-HEX


     IS THE ABOVE OK ?   >YES


          * SPACE BAR MOVES CURSOR *
         * ARROW KEYS CHANGE VALUES *
       * RETURN CONCLUDES DATA ENTRY *
           * ESCAPE EXITS PROGRAM *
```

FIGURE 3.2 — INIT SCREEN WHEN READY TO WRITE

If you discover an incorrect entry, use an arrow key to change the YES value to NO and press the RETURN key. You will find the cursor at the top line. Using the space bar, move the cursor to the line containing the incorrect value and change it using the arrow keys. Even if you have not made any mistakes you may wish to try this to see how easily errors can be corrected. When your screen matches that of Figure 3.2, simply press the RETURN key. You will once again see the question "IS THE ABOVE OK ?" followed by the answer YES. This time press the RETURN key to indicate that the above data is indeed correct.

Next you will see a four line prompt:

```
        EXISTING DATA WILL BE OVERWRITTEN
       INSERT DISKETTE IN DRIVE 01 SLOT 06
           PRESS RETURN TO CONTINUE
            PRESS ESCAPE TO ABORT
```

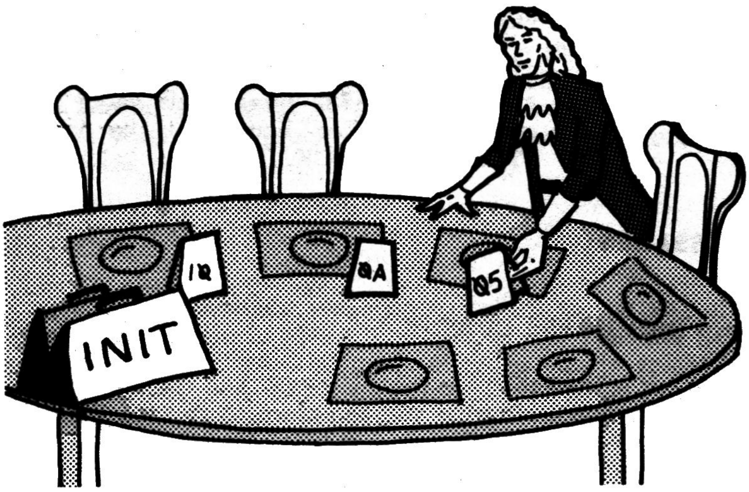
After making sure the blank diskette is in the slot and drive indicated and that the drive door is closed, you may press the RETURN key to begin formatting. Should you discover an error at this point, you may use the ESC key to abort. You would be returned to the top of the screen at which time you could make corrections. You will hear the disk arm recalibrate the same way it does when you perform a normal DOS "INIT" command. Shortly thereafter you will see a prompt line indicating the number of the track being initialized. The whole procedure should take less than 25 seconds, at which time you will have a formatted diskette. This diskette does not contain a DOS boot image, as would a diskette initialized normally, nor does it contain any programs (HELLO or otherwise). The VTOC (Volume Table of Contents) has been modified to allow data to be stored on tracks 1 and 2 (normally reserved for the DOS boot image) thereby providing 32 extra sectors. You may, of course, place DOS on the diskette using the appropriate program on your System master diskette (such as MASTER CREATE). If you do this, be sure to use the FIXCAT utility (Chapter 5) to mark tracks 1 and 2 of the diskette in use.

When the formatting is complete, you should see the prompt line "EXIT PROGRAM?". This concludes the INIT tutorial so you may select your answer (using the arrow keys) and then press RETURN.

INIT OPTIONS

DISK SECTORING	This value is the number of sectors per track. The options are 13 or 16. A 16 sector selection will not work if you have an old disk drive which has not been updated for 16 sectors per track (the P6A ROM must be installed on the controller card).

DISK FORMAT	This is the type of operating system being used on this diskette. For 13 sectors both DOS and CP/M are available. For 16 sector diskettes DOS, CP/M and PASCAL are available (the PASCAL option works for APPLE FORTRAN). Our purpose in supporting PASCAL and CP/M in INIT was primarily intended to provide the ability to reinitialize a track which might have one or more errors in it. While it is possible to initialize an entire diskette without preserving data using the standard PASCAL or CP/M skewing, the resulting diskette may be missing special formatting required by the operating system. Therefore we do not recommend using INIT to initialize diskettes for either PASCAL or CP/M.
PRESERVE DATA	This question asks whether the data currently on your diskette should be preserved. The options are YES or NO. If you answer YES, each sector on a track will be read into memory, the track will be reformatted, the original data rewritten to the track. Note that this constitutes an "INIT in place" and can result in the loss of an entire track's worth of data if your diskette is physically damaged and cannot be reformatted. For this reason, we recommend that you make a backup before using INIT to preserve data; see the Appendix to this manual. If you answer NO, existing data on the diskette will not be preserved and zeros will be written to each newly formatted sector.
SKEW DIRECTION	This prompt indicates the direction of the skewing to be used. The options are ASCENDING and DESCENDING. Ordinarily, DOS reads sectors in DESCENDING order while CP/M and PASCAL read them in ASCENDING order.



SPECIFYING SECTOR ORDER

SKEW FACTOR	This is the spacing placed between logically sequential sectors during formatting. For 13 sector diskettes the values 1-12 are available and for 16 sectors the values 1-15. Standard DOS diskettes are skewed with 2 DESCENDING. See the section, HOW SECTOR SKEWING CAN AFFECT DISK PERFORMANCE for more details on this option.
SLOT	The number of the slot occupied by your Disk II disk drive controller card. Slots 1, 2, 3, 4, 5, 6, and 7 are supported.
DRIVE	The drive number of your disk drive. The options are 1 or 2.
VOLUME NUMBER	This is the volume number that will be used to format your diskette. The numbers 0-254 ($00-$FE Hex) are available. Additionally a DEFAULT option is available which will use the current volume number of your diskette. If none is found or you are not preserving data, the value 254 ($FE Hex) will be used.

STARTING TRACK	The track number upon which formatting is to start. The options are 0-34 ($00-$22 Hex).
ENDING TRACK	The last track to be formatted. The options are 0-34 ($00-$22 Hex). This value must be greater than or equal to the value for STARTING TRACK.

INIT MESSAGES

* SPACE BAR MOVES CURSOR *

The space bar is used to move the cursor (>) to the next line. The cursor wraps around, allowing you to return to the top line.

* ARROW KEYS CHANGE VALUES *

The arrow keys (← and →) are used to change data values. A wrap around feature allows all possible values to be accessed with either arrow key.

* RETURN CONCLUDES DATA ENTRY *

The return key indicates that data entry has been concluded. Press RETURN when you are sure that all the entries have been set to the correct values for the particular task you are performing.

* ESCAPE EXITS PROGRAM *

The escape key (ESC) allows you to exit the program. As a safeguard against hitting escape accidentally, you will see the message EXIT PROGRAM ? > YES. If you do not wish to exit the program change the answer to NO with either arrow key and press RETURN. Otherwise simply press RETURN and you will be returned to the Bag of Tricks main menu.

```
IS THE ABOVE OK ?
```

This prompt asks whether the information in the data entry area is correct. The options are YES and NO. NO will return the cursor to the top line of the screen so that the data can be modified. YES causes the program to proceed using the input parameters as they appear on the screen.

```
EXIT PROGRAM ?
```

This prompt asks whether you wish to return to the Bag of Tricks main menu or remain in INIT. The options are YES and NO. YES will return you to the initial menu allowing you to select another Bag of Tricks program. NO will rerun the INIT program so that you may continue formatting diskettes.

```
EXISTING DATA WILL BE OVERWRITTEN
INSERT DISKETTE IN DRIVE XX SLOT XX
PRESS RETURN TO PROCEED
PRESS ESCAPE TO ABORT
```

This series of messages occurs when you are initializing a diskette without preserving data. The message indicates which slot and drive are to be used. Any data now existing on the indicated diskette will be lost. The return key will start the initialization process. The escape key allows you to abort the process and return to the data entry area.

```
DATA WILL BE PRESERVED
INSERT DISKETTE IN DRIVE XX SLOT XX
PRESS RETURN TO PROCEED
PRESS ESCAPE TO ABORT
```

This series of messages occurs when you are preserving data on a diskette while reformatting one or more tracks. Any data found on the diskette will be written back after each track has been initialized. The return key will start the reinitialization process. The escape key allows you to abort the process and return to the data entry area.

INITIALIZING TRACK XX-DEC XX-HEX

During initialization this message indicates the track currently being formatted.

BUILDING CATALOG 17-DEC 11-HEX

This message occurs only when formatting track 17 ($11 Hex) during an initialization employing DOS format. Both an empty catalog and a VTOC (Volume Table of Contents) are constructed at this time. INIT assumes that no DOS boot image is placed on the diskette, and the new VTOC is marked to indicate this.

READING TRACK XX-DEC XX-HEX

This message occurs when INIT is reading data from the indicated track. Any valid sectors found will be stored in a buffer area for later rewrite to the newly formatted track.

WRITING TRACK XX-DEC XX-HEX

This message occurs when INIT is writing data to the indicated track. After first reformatting the track, the data previously stored in the buffer area is written back to the diskette.

```
* * DISKETTE WRITE PROTECTED * *
CTRL-P TO PROCEED
CTRL-A TO ABORT
```

This series of messages indicates that the write protect switch in your disk drive is closed. Most likely this means that the notch on your diskette has been covered. You must remove the write protect tab from your diskette if you wish to write to it.

```
* * XX OF XX SECTORS UNREADABLE * *
CTRL-P TO PROCEED, IGNORE READ ERRORS
CTRL-A TO ABORT
```

This series of messages tells you how many sectors could not be preserved on the indicated track. If you press CTRL-P to proceed, the unreadable sectors will be replaced with zeroed sectors on the newly formatted track.

```
* * UNABLE TO FORMAT * *
CTRL-P TO PROCEED
CTRL-A TO ABORT
```

This series of messages usually indicates that there is something physically wrong with the diskette itself. Check to make sure that there is actually a diskette in the appropriate disk drive and that the drive door is closed. If this message appears while data is being preserved, any data in INIT's buffer (preserved sectors from the damaged track) has been lost. To avoid this possibility it is recommended that you first make a backup copy of a suspected damaged diskette before attempting to fix the diskette using INIT. See the Appendix of this manual for a method of backing up damaged diskettes.

HOW SECTOR SKEWING CAN AFFECT DISK PERFORMANCE

Sector skewing, or interleaving, is a method by which the time required to access a diskette sector can be reduced. Whenever you initialize a diskette, either by using the DOS INIT command or by using the COPY program to create a new diskette, DOS formats each track by arranging the sectors in a "physical" order. Each physical sector number is written in the Sector Address Field (see Beneath Apple DOS, Chapter 3). In DOS 3.3 this order is sequential, from physical sector zero to physical sector fifteen. When DOS 3.3 reads a sector it translates the "physical sector number" into a "logical sector number" using a soft skewing table, located in the Read/Write Track/Sector portion of DOS. Most programs and the DOS file manager refer to sectors by their logical sector number. When DOS 3.3 was written, the soft skewing table was established so that the logical order in which sectors were arranged on a track could be controlled merely by modifying the soft skew table. (Earlier versions of DOS set up sector skewing when the physical sectors were placed on the diskette at INIT or COPY time — in fact, under DOS 3.2 and DOS 3.1, INIT and COPY each used a different skew pattern!) A diagram of the process used by DOS 3.3 to reference sectors is shown in Figure 3.3.

Why is sector skewing important? To understand skewing it is enlightening to look at the series of events which occur whenever a sector is read from or written to the diskette. Figures 3.4 through 3.6 will be used to describe what happens when a program is read from a standard DOS diskette. To simplify this discussion, these three figures will show the **logical** sector numbers on the "diskette". In actual fact, the physical order is that of Figure 3.3.

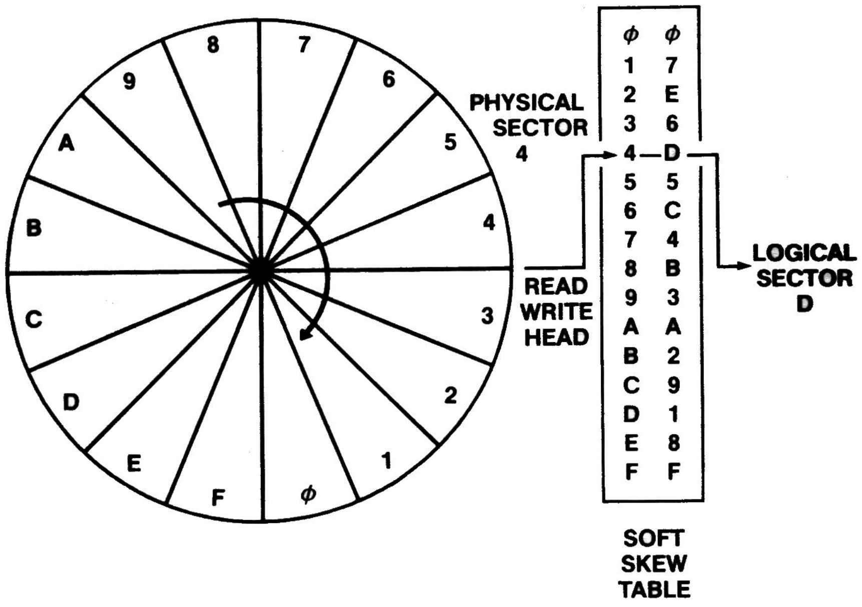


FIGURE 3.3 — PHYSICAL SECTOR ORDER

Shown in Figure 3.4 is the standard skewing for DOS 3.3. There are different skews for PASCAL/FORTRAN and CP/M. The reason for this should become apparent as this discussion proceeds. When DOS 3.3 was designed, a skewing was chosen that optimizes access during booting. Since sector reads occur fairly rapidly during the boot process and sectors are read in reverse order (sector F first, then E, then D, etc.), a "2 DESCENDING" skew was chosen. This means that the next lower sector is (nearly) always two sectors away from the last one. For example, sector 6 is two sectors beyond sector 7. Although this standard DOS skewing allows disks to boot in at optimal speed, it turns out to be a very poor skewing for doing almost anything else, like loading or running programs (BLOAD, LOAD, BRUN, RUN).

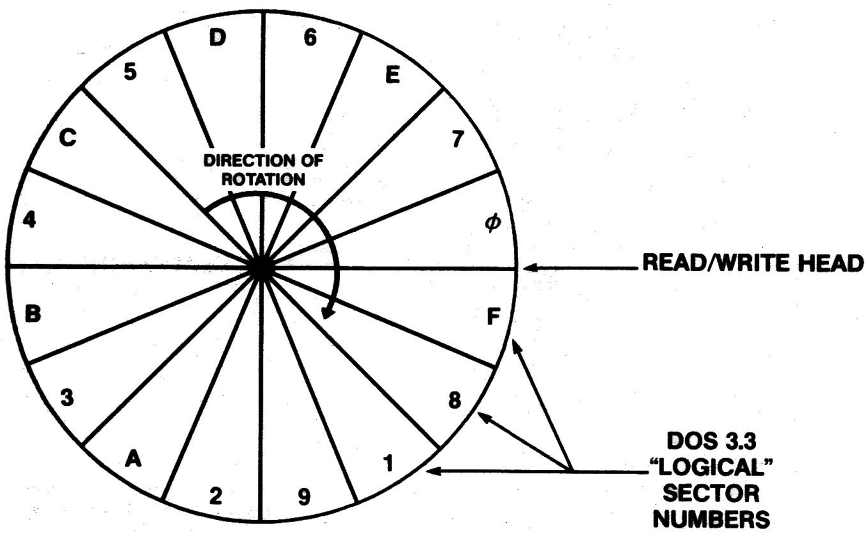


FIGURE 3.4 — LOGICAL SECTOR ORDER, STANDARD DOS 3.3

Referring again to Figure 3.4, notice that the read/write head is positioned where it would be immediately after reading Sector F. At this point, control returns from RWTS and the file manager processes the data it has just read, determines which sector must be read next, and calls RWTS again. At the same time all of this is going on the diskette continues to spin in the drive. Thus, several sectors pass beneath the read/write head before the file manager is ready to request another read. In the usual case, the file manager will want to read the next lowest numbered sector on the track (sector E, in this case), and this sector will be one of those which has already passed by. Figure 3.5 shows the position of the diskette relative to the read/write head when the file manager is ready to read sector E.

Notice that one of the "passed over" sectors was, in fact, sector E. RWTS will now read each sector on the diskette until it again finds sector E. The period of time it takes to find the desired sector after the read has been requested is referred to here as the "rotational delay" for the disk access. Figure 3.6 shows the read/write head in position to read sector E and identifies the rotational delay. This delay is wasted time.

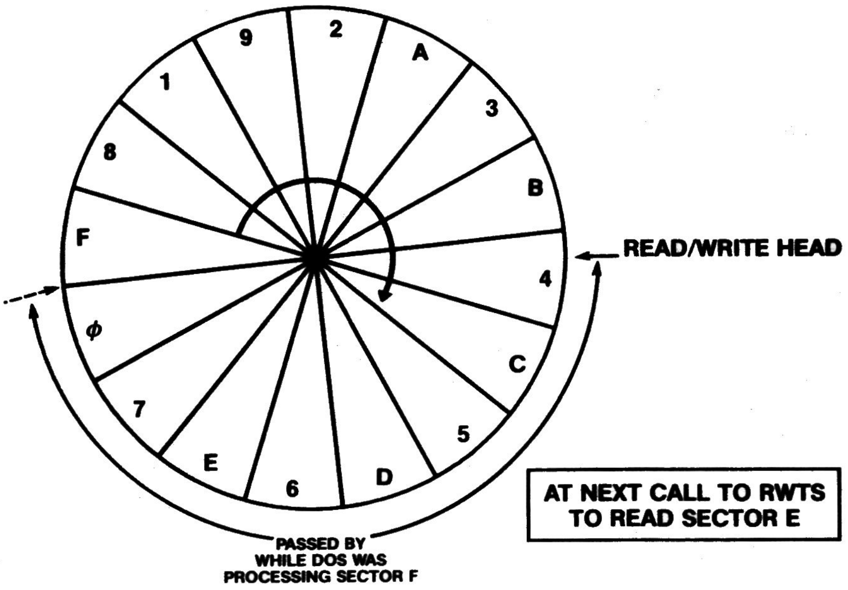

9
2
A
3
B
4
C
5
D
6
E
7
ϕ
F
8
1
READ/WRITE HEAD
AT NEXT CALL TO RWTS TO READ SECTOR E
PASSED BY WHILE DOS WAS PROCESSING SECTOR F

FIGURE 3.5 — STANDARD DISKETTE POSITION AFTER PROCESSING SECTOR F

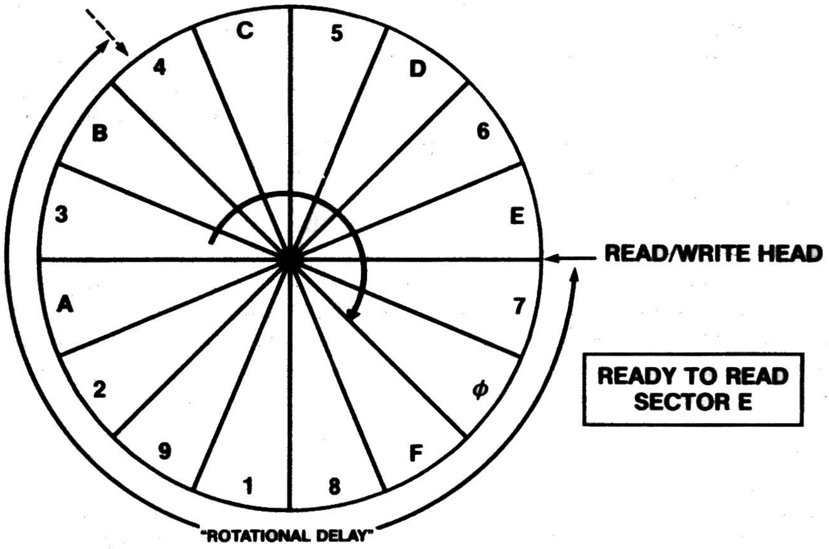


FIGURE 3.6 — STANDARD DISKETTE POSITION AFTER ROTATIONAL DELAY

You may have noticed that, had the file manager processed sector F faster, letting only one sector pass by in the meantime, it could have returned to the disk in time to read sector E as it passed under the read/write head. This is apparently what the DOS designers intended and, in fact, this is what occurs during the boot process. The example given above, however, was for a BLOAD operation (BLOAD, LOAD, BRUN, and RUN have similar patterns of behavior, using identical code within DOS).

Obviously, unless you spend a lot of time booting disks, it would be nice to reduce the rotational delay for other operations as well. One solution would be to change the soft skew table in RWTS to a more appropriate pattern. This works but has two major disadvantages. One disadvantage is that once this table is changed, any diskette created using the standard skew table is not accessible using the modified soft skew table, and vice versa. For example, what standard DOS thinks is sector 7, the modified DOS may think is sector 5. A second disadvantage is that this scheme applies to every track on the diskette, including the boot image of DOS on tracks 0, 1, and 2, making a BLOAD command run faster at the expense of the time to boot the disk.

Luckily, there is another way to deal with the problem. Instead of changing the soft skew table, the sectors can be physically rearranged on the track such that, when run through DOS's boot-optimized skew table, they will be in the optimal pattern for LOAD and BLOAD. And it is possible to have different physical arrangements on different tracks. Thus, tracks 3 through 34 can be optimized for LOADing while the boot tracks are optimized for booting. Also, disks created in this way may be read by any DOS (of the appropriate version).

It turns out that during a BLOAD operation the file manager is "out to lunch" processing the previously read sector for about the time it takes eight sectors to pass beneath the read/write head. Thus a "9 DESCENDING" skew seems a good choice (9 DESCENDING provides eight sectors of padding between descending sequential sectors). Figure 3.7 shows the logical arrangement of sectors on the diskette when a 9 DESCENDING skew is used.

Using this new skewing, after the file manager has finished processing sector F, the read/write head is positioned very near to sector E. Figure 3.8 shows the position of the diskette at this time.

The actual rotational delay is less than one sector long, as shown in Figure 3.9.

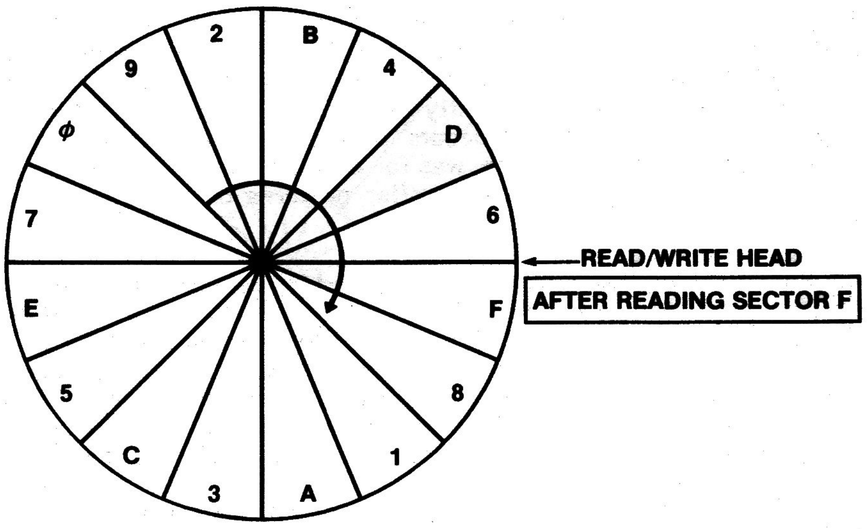


FIGURE 3.7 — LOGICAL ORDER FOR A "9 DESCENDING" SKEW

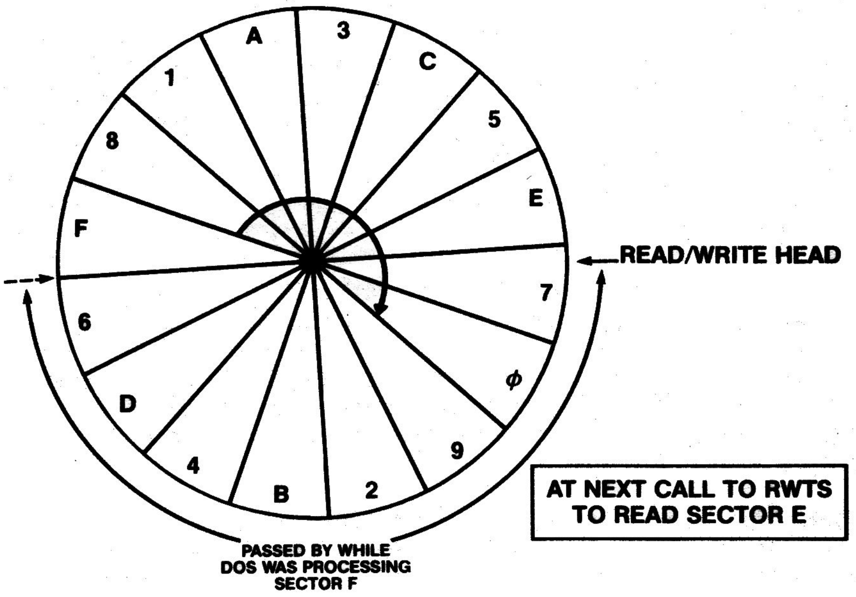

A
3
C
5
E
7
φ
9
2
B
4
D
6
F
8
1
READ/WRITE HEAD
AT NEXT CALL TO RWTS TO READ SECTOR E
PASSED BY WHILE DOS WAS PROCESSING SECTOR F

FIGURE 3.8 — 9 DESCENDING SKEW AFTER PROCESSING SECTOR F

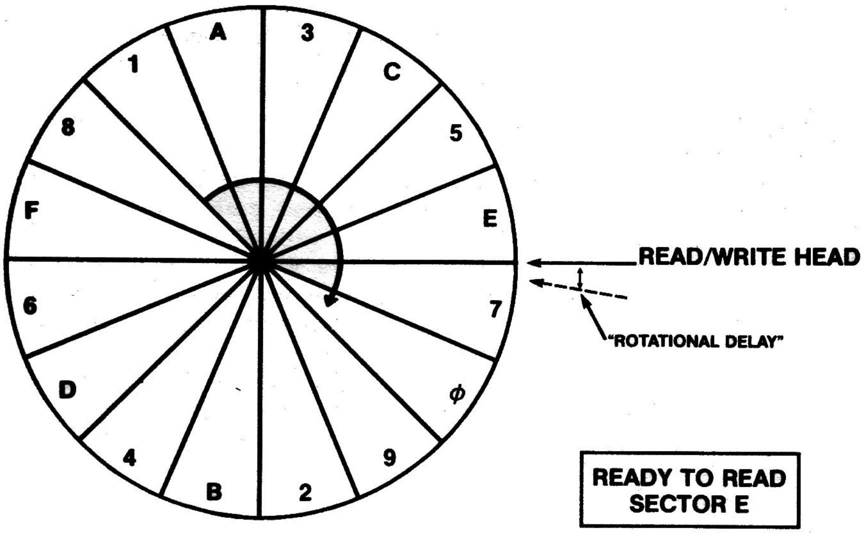


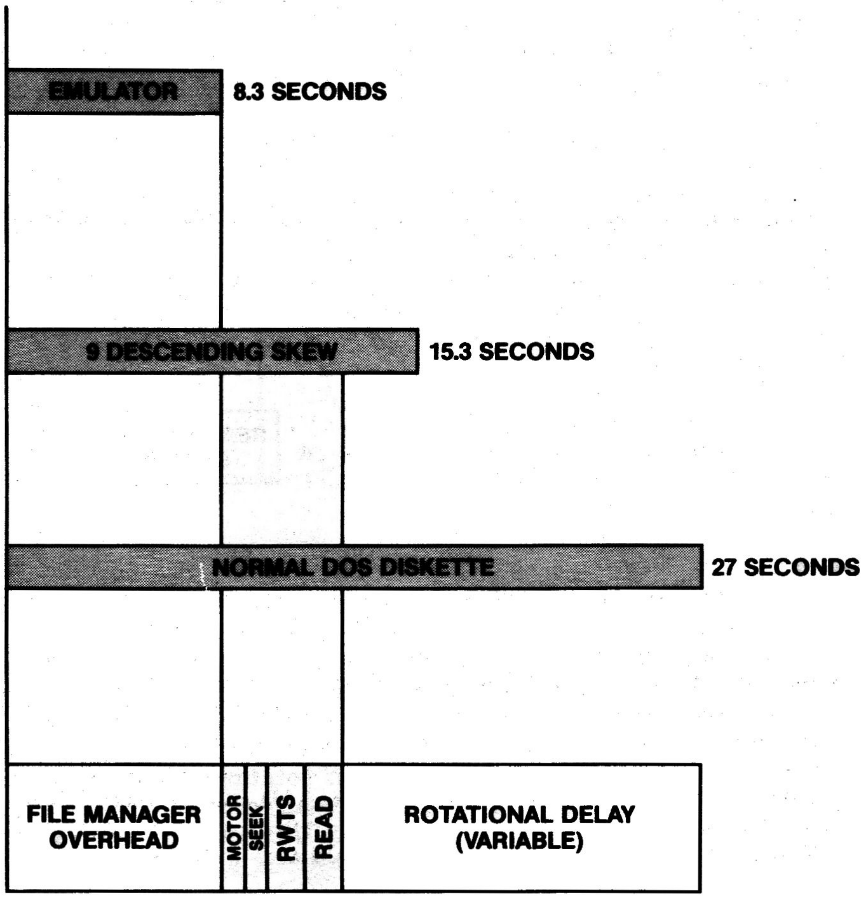
FIGURE 3.9 — 9 DESCENDING SKEW AFTER ROTATIONAL DELAY

One might ask just how significant a contribution rotational delay makes to the overall access time of a disk sector. Within about a ten percent tolerance, a DISK II spins at 300 revolutions per minute. This means that every sector on the track passes beneath the read/write head 300 times per minute. If rotational delay amounts to waiting for ten sectors to go by before reading the desired sector, as in the standard DOS skewing with BLOAD, and 16 sectors are read on every track, the time wasted per track amounts to:

$$\frac{1}{300}\ \frac{\text{min}}{\text{rev}} \times 60\ \frac{\text{sec}}{\text{min}} \times 10\ \frac{\text{lost sectors}}{\text{read}} \times 16\ \frac{\text{reads}}{\text{track}} \times \frac{1\ \ \text{rev}}{16\ \text{sector}}$$

= 2 lost seconds per track

This value is theoretical and is not completely accurate, since rotational delay varies depending upon factors such as random placement after moving to a new track, variations in DOS response, etc. However, experimental measurements have shown it to be within 10 per cent of the actual delay in most situations. If an Applesoft program which is stored in 64 sectors on the diskette is loaded, this rotational delay amounts to eight seconds. With the 9 DESCENDING skew this delay is reduced to less than half a second!



MOTOR	**— Delay waiting for disk motor to come up to speed**
SEEK	**— Time spent moving disk arm from track to track**
RWTS	**— Read/Write Track/Sector overhead: Postnibble etc.**
READ	**— Time spent reading data sectors**

FIGURE 3.10 COMPARISON OF TIMES TO BLOAD A FILE OF $7000 BYTES

Obviously, the more sectors accessed the greater the rotational delay. The chart in Figure 3.10 shows the times required to BLOAD a binary program which is $7000 bytes in length ($70 sectors or 112 decimal). Using both experimental and theoretical methods, the component delays to a complete disk access have been found.

Figure 3.10 shows that rotational delay amounts to more than 50 per cent of the time required to load the file with standard DOS skewing! By reskewing to 9 DESCENDING an improvement in disk access time of 43 per cent was realized. Another interesting point is that the DOS file manager overhead is quite high. This is the processing time required to execute the instructions in the file manager program itself, and this time is required even by the commercially available "disk emulators" which simulate a disk drive using RAM. Thus, emulators can do no better than to cut the delay back to the file manager overhead as a minimum. This effects about a three times improvement over standard DOS but not even a two times improvement over the optimal skew. By reskewing, with no additional investment in hardware, we have gained almost 2/3 of the advantage of a disk emulator!

One further word on skewing as it applies to non-program files is in order. The pattern of access of a BASIC program to its data files varies drastically from program to program. Many times an application will not return to the disk to read the next sector for many revolutions — it might even allow the drive to turn itself off! In this case, sector skewing becomes a consideration of relatively low importance. Since access can also be affected by the number of records in each sector, it becomes almost impossible to identify a repeating computational delay. The best that can be said is that it does not matter much which skew is used for data files and 9 DESCENDING will probably work as well as any other.

The table in Figure 3.11 shows the difference between the standard 2 DESCENDING skew and a 9 DESCENDING skew for typical disk activities. It should be noted that these values may vary, because a number of variables can affect the outcome, including hardware and even the brand of diskette used. For this reason the reader is encouraged to experiment with different skews, using the Bag of Tricks INIT program, to find the best one for his particular application.

Time in Seconds

Activity	2 Desc. Skew	9 Desc. Skew	% Difference
Boot	7.0	6.6	6
Boot/load lang card	19.5	13.3	32
LOAD Basic File	8.0	5.0	38
SAVE Basic File	12.7	11.3	11
BSAVE X,A$800,L$7FFF	44.2	39.6	10
BLOAD X	31.7	17.4	45
Read Text File	7.9	8.4	- 6
Write Text File	10.2	9.0	12
CATALOG	2.7	2.6	- 4

FIGURE 3.11 — TIMING FOR TYPICAL DISK ACTIVITIES

Whereas the normal skewing on a DOS 3.3 diskette is far from optimal, both PASCAL and CP/M appear to be optimally skewed. PASCAL, which reads a 512 byte block (two Apple sectors), is skewed 2 ASCENDING. This results in the most efficient read possible for a block. CP/M reads a 1024 byte allocation unit (four Apple sectors) and is skewed 3 ASCENDING. This also seems to provide the greatest efficiency demonstrating that in both cases great care was taken to insure optimal disk access. We recommend using these standard skewing offsets for normal PASCAL and CP/M operations. For particular applications, however, you should feel free to experiment with alternate skewing patterns.

Regarding DOS 3.2, we have not made exhaustive tests. However, current INIT and COPY programs use a 9 DESCENDING skew. A 6 DESCENDING skew seems to be optimal for loading programs under DOS 3.2, and a 2 ASCENDING skew on tracks 0-2 is optimal for booting.

CHAPTER 4
ZAP — By Don Worth

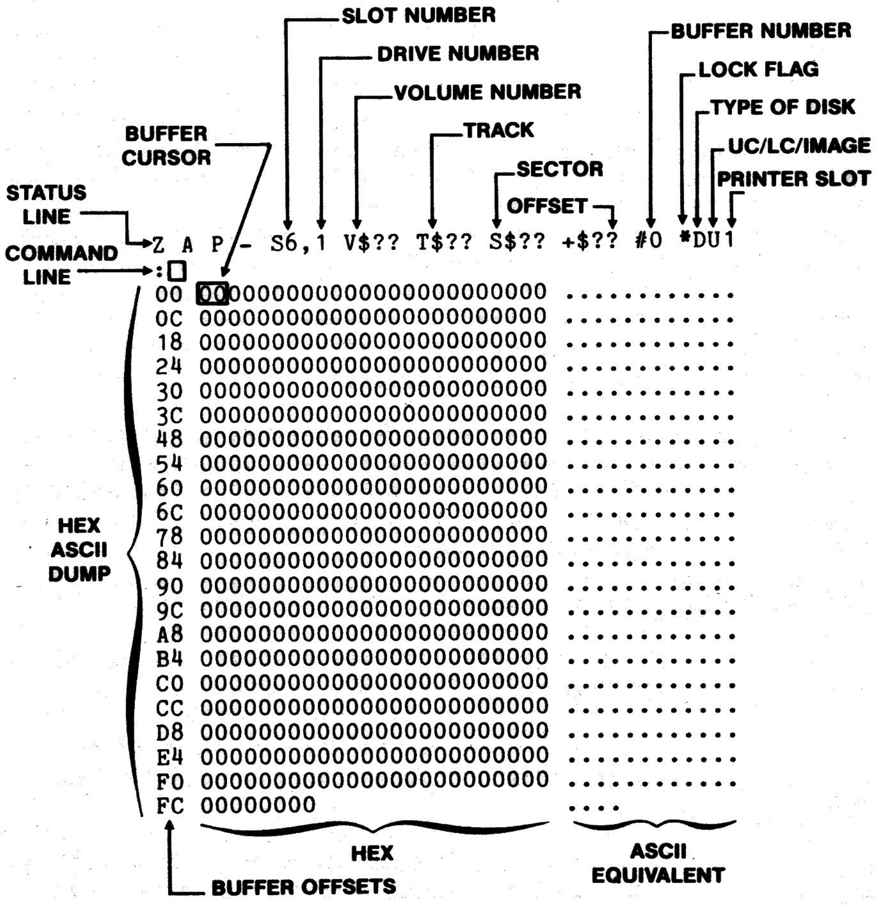
The ZAP utility allows you to read selected sectors from a diskette and examine or modify them. It was originally patterned after the IBM SUPERZAP utility but it is far more interactive. ZAP can be used to examine normal, unprotected diskettes of both 13 and 16 sector formats. In addition, ZAP allows you to access files on DOS diskettes as well as CP/M and PASCAL 16 sector diskettes. Although ZAP provides over 50 commands and is extremely powerful, a small subset of its commands can be used by the novice to perform almost any operation desired.

For the user's protection, ZAP defaults to a write protected mode. This means that no matter what you do while experimenting with ZAP, you can not write to or otherwise damage your diskette. To leave the write protected mode you must issue the ZAP command, UNLOCK. This feature is intended to assist novices while they are learning ZAP as well as to provide an added security level against accidental damage to diskettes.

A BRIEF ZAP TUTORIAL

At this point a demonstration is in order. Boot up the Bag of Tricks diskette, invoke ZAP, and follow along with the tutorial.

Upon entry to ZAP you will see a screen which looks like Figure 4.1. Most of the time you are in ZAP you will see screens similar to this one.

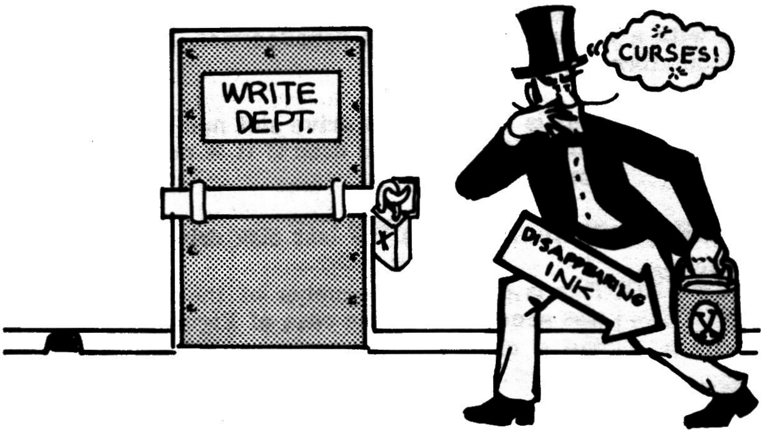


```
Z A P - S6,1 V$?? T$?? S$?? +$?? #0 *DU1
:
00 000000000000000000000000 ............
0C 000000000000000000000000 ............
18 000000000000000000000000 ............
24 000000000000000000000000 ............
30 000000000000000000000000 ............
3C 000000000000000000000000 ............
48 000000000000000000000000 ............
54 000000000000000000000000 ............
60 000000000000000000000000 ............
6C 000000000000000000000000 ............
78 000000000000000000000000 ............
84 000000000000000000000000 ............
90 000000000000000000000000 ............
9C 000000000000000000000000 ............
A8 000000000000000000000000 ............
B4 000000000000000000000000 ............
C0 000000000000000000000000 ............
CC 000000000000000000000000 ............
D8 000000000000000000000000 ............
E4 000000000000000000000000 ............
F0 000000000000000000000000 ............
FC 00000000                 ....
```

FIGURE 4.1 — BEGINNING ZAP DISPLAY

The top line of Figure 4.1 is an abbreviated status readout. Values given with a dollar sign ($) are in hexadecimal. From left to right the indicators are as follows:

SLOT NO.	This is the last disk drive slot number accessed and it will be the next one used if it is not changed by an S command.
DRIVE NO.	Similarly, this is the last disk drive number used.
VOLUME NO.	This is the volume number of the last diskette which was accessed. In the case of Figure 4.1, ZAP has not had a chance to access any diskette so no volume number is yet available.
TRACK	The last track accessed is displayed here. No track has yet been accessed.
SECTOR	The last sector accessed. Taken together, the TRACK and SECTOR readout indicate the origin of the data displayed in the hex/ASCII display below.
OFFSET	This is the current byte offset into the sector buffer, given in hex. It indicates the current position of the buffer cursor (to be described later) within the buffer. At present, since no data has yet been read from the disk, it has no value.
BUFFER NO.	There are 16 buffers (memory areas into which you can read data) provided by ZAP. They are numbered $00 through $0F. At present, buffer $00 is selected and displayed. The buffer number indicator is also used to remind you that you have made changes to the buffer. If you should modify the buffer contents, the buffer number will be displayed in inverse video.
LOCK FLAG	Whenever you first enter ZAP, it places itself in LOCK mode. As mentioned earlier, this prevents you from accidentally writing on your diskette. When you are in LOCK mode, an asterisk (*) appears in this position on the screen.



ZAP DEFAULTS TO THE LOCKED MODE
TO PREVENT ACCIDENTAL WRITING

TYPE OF DISK	Initially, ZAP assumes you want to examine a 16 sector DOS diskette. A "D" is displayed here to indicate this. If you issue the appropriate commands you can change this setting to PASCAL ("P") or CP/M ("C"). Both 13 and 16 sector DOS disk types are denoted by a "D". To determine which is in use, use the STATUS command (described later).
UC/LC/IMAGE	The character printed here indicates which translation is used on the ASCII display given below on the right side of the screen. If a standard ASCII translation is used, a "U" or a "L" is printed to indicate whether lower case characters are printed as their upper case equivalent ("U") or printed as lower case ("L"). If IMAGE mode is in effect (described later), an "I" appears here.
PRINTER SLOT	This is the slot number for an optional printer. If you have a printer controller card in the indicated slot, you may issue commands within ZAP to dump the sector in hex and ASCII onto the printer, create an audit trail of changes you make to the disk, etc. When LOGGING (audit trail) is turned on, the printer slot number is printed here in inverse video.

Just below the status line is the command input line. It is here that you type commands for ZAP. Notice the cursor (the familiar flashing block) next to the prompt character (a colon). The right half of the command line is sometimes used to display error messages as well.

Below the command input line is the hexadecimal and ASCII display of the contents of the current sector buffer. In the example of Figure 4.1, no sector has been read so zeros are displayed. Given at the far left are the hex offset locations for the first byte on each line. This is done so that you can easily determine the offset of any byte by counting to the right from the first on its line. Each line (except for the last one) shows 12 bytes of hex followed by its equivalent in printable ASCII. If this were the display of the catalog, for example, you would see file names over on the right hand side of the screen. A total of 256 bytes (one sector's worth) is displayed.

Notice that the first hexadecimal byte is in inverse video. This indicates that the buffer cursor is pointing to this byte. Using ZAP commands, you may move the buffer cursor around the buffer to indicate where changes are to be made.

At this point you may be wondering about the concept of a sector buffer. In general, ZAP allows you to read a sector into the sector buffer (which is really just a portion of memory at $5000 in your machine). When this happens, the sector is **copied** from the diskette into this 256 byte memory area. Nothing is changed on the diskette itself. Using ZAP commands, you may change the contents of this buffer to your heart's content. You have not changed the original sector on the diskette in any way. Should you wish to change the sector, you must instruct ZAP to write the contents of the sector buffer back over some sector on the diskette (you don't have to write it over the same sector from which it was read if you don't want to but this is usual). Working with the buffer in this way, you can take your time making corrections on a sector by sector basis, reducing your chances of making mistakes that ruin your diskette.

Now it is time to actually read a sector from the diskette into the buffer. Insert your practice diskette into your drive and type the following ZAP command, followed by a carriage return.

```
R11,0
```

(If your practice diskette is formatted with 13 sectors you should first type the command, DOS13, to inform ZAP of this.) The disk drive will turn on and in a moment the display will change. What you have just entered is a READ command, instructing ZAP to read the sector it finds on track $11, sector $00. If you have read **Beneath Apple DOS** you know that this is the Volume Table of Contents (VTOC) for the diskette. The actual contents of the VTOC for your practice diskette are displayed in hex and ASCII on the screen.

Now you will tell ZAP to read the first CATALOG sector. The CATALOG resides on the same track as the VTOC so all you need to do is specify the sector number you want — you need not respecify the track:

```
        R,F
```

```
Z A P - S6,1 V$01 T$11 S$0F +$00 #0 *DU1
:R,F
00 00110E000000000000000012 ............
0C 0F02C8C5CCCCCFA0A0A0A0A0 ..HELLO
18 A0A0A0A0A0A0A0A0A0A0A0A0
24 A0A0A0A0A0A0A00200130F       ....
30 04C2C9CEC1D2D9A0C6C9CCC5 .BINARY FILE
3C A0A0A0A0A0A0A0A0A0A0A0A0
48 A0A0A0A0A0A00200140F02      .....
54 C1D0D0CCC5D3CFC6D4A0D0D2 APPLESOFT PR
60 CFC7D2C1CDA0A0A0A0A0A0A0 OGRAM
6C A0A0A0A0A00200150F00D4       .....T
78 C5D8D4A0C6C9CCC5A0A0A0A0 EXT FILE
84 A0A0A0A0A0A0A0A0A0A0A0A0
90 A0A0A0A0A002000000000000      .......
9C 000000000000000000000000 ............
A8 000000000000000000000000 ............
B4 000000000000000000000000 ............
C0 000000000000000000000000 ............
CC 000000000000000000000000 ............
D8 000000000000000000000000 ............
E4 000000000000000000000000 ............
F0 000000000000000000000000 ............
FC 00000000                 ....
```

FIGURE 4.2 — A CATALOG SECTOR

Here you are telling ZAP to read sector $0F on the current track (track $11). (If your practice diskette was made by DOS 3.2 or DOS 3.1 you should type "R,C" instead, since sector $0C is the last sector of a 13 sector track.) You should now be looking at the first sector of the CATALOG. Your screen should look like Figure 4.2.

Suppose you wanted to use ZAP to change the name of one of the files in the CATALOG. This may not sound too useful, since the DOS RENAME command will do this. Under some circumstances in DOS it is possible to create a file with a control character imbedded in the name, making it difficult if not impossible to delete or rename the file. To correct such a situation, it might be necessary to use ZAP on the CATALOG directly. Imagine that the file named "HELLO" has a bad character in it. To fix the name, you must position the buffer cursor to the part of the buffer you want to modify. Type the following command:

```
        +E
```

This tells ZAP to position the buffer cursor forward $0E (14 decimal) bytes from its current position ($00). The cursor should now be over a $C8, the "H" in HELLO. Now type:

```
        :'HI THERE'
```

(Be sure to type the colon) You have now told ZAP to store the string 'HI THERE' into the buffer, starting at the buffer cursor. Notice that ZAP has advanced the cursor to point past the last character stored. At this point you might think you are done. You have renamed the file to HI THERE. Wrong! Remember that all you have done so far is to update the memory image of the sector in the buffer — you have not yet written that updated image back to the diskette. ZAP reminds you that you have modified the data originally read in by highlighting the buffer number on the status line. Let's pretend that you have forgotten to rewrite the modified sector, and try to read the VTOC again. Type in the command:

```
        R,0
```

What happens? ZAP beeps and prints an error message:

```
        WARNING: BUFFER CHANGED
```

Notice that the buffer number is not highlighted anymore. This means you will only get one warning. If you do not write the buffer now, but rather re-enter the read command, ZAP will give up and do the read, discarding the changes you have made to the buffer. Instead, heed the warning and write the sector back to disk. Enter the command:

```
WRITE
```

What you get is another error message!

```
WRITE PROTECTED
```

Remember that LOCK protection is assumed by ZAP. You must UNLOCK ZAP before it will allow you to do anything which might change your diskette. Do this now:

```
UNLOCK
```

Notice that the asterisk (*) has disappeared from the status line. ZAP will now permit write commands. Try the WRITE command again. This time it works! The drive turns on for a moment and then stops. Before proceeding, relock ZAP by typing LOCK. To convince yourself that the change has been made, re-read the sector by typing:

```
R
```

No track or sector number was given since ZAP will assume you mean to read the current sector. (The R command is handy when you want to cancel some modifications you have made to the buffer). In this case nothing changed on the screen but the drive turned on and off. This means that what was read matches exactly what is in the buffer. The change must have been made on the diskette. You can verify this later by CATALOGing the disk under DOS.

Now we will try some other features of ZAP. It turns out you don't need to remember the location of the VTOC or CATALOG. ZAP knows where they are and will find them for you if you enter either the VTOC or the CAT command. Try it!

Up to now you have made absolute references to the diskette's tracks and sectors. ZAP will also let you "open" a file. By file we mean one of the files catalogued by the DOS file manager. While the file is open, all references are made as "file relative". This means that, instead of asking ZAP to read track $11, sector $00, for instance, you will ask it to read the $04th sector of the file (or even the $57th record, if you are looking at a random text file). You can try this out on a very short file by opening one of the files on your practice diskette. Type:

OPEN'BINARY FILE'

The disk turns on and after a while the screen looks like Figure 4.3.

```
Z A P - S6,1 V$01 RSA=$0000 +$00 #0 *DU1
:OPEN'BINARY FILE'
00 D0033004CBF9D4C849D4CFD P.O.L?.L..L
0C AA4CB5B7AD0F9DAC0E9D60AD *L57-..,..,-
18 C2AAACC1AA604C51A8EAEA4C B*,A*,LQ(JJL
24 59FABF9D384C58FF4C65FF4C YZ?.8LX_LE_L
30 65FF65FFA000000000000000 E_E_ .......
3C 000000000000000000000000 ............
48 000000000000000000000000 ............
54 000000000000000000000000 ............
60 000000000000000000000000 ............
6C 000000000000000000000000 ............
78 000000000000000000000000 ............
84 000000000000000000000000 ............
90 000000000000000000000000 ............
9C 000000000000000000000000 ............
A8 000000000000000000000000 ............
B4 000000000000000000000000 ............
C0 000000000000000000000000 ............
CC 000000000000000000000000 ............
D8 000000000000000000000000 ............
E4 000000000000000000000000 ............
F0 000000000000000000000000 ............
FC 00000000                 ....
```

FIGURE 4.3 — A BINARY FILE SECTOR

Notice that the status line has changed somewhat. The T$ and S$ (track and sector) have been replaced with a relative sector address (RSA). You are looking at the $00th relative sector of the file. The next sector, if there was one, might be at the other end of the diskette, but it could be accessed now by typing R1. You will not be able to read absolute track/sectors while the file is open, only sectors within the open file. You can always determine whether a file is open by looking at the status line for the T and S or the RSA.

While you are looking at the first sector of a typical binary file, you can find out the Address and Length values for the file by looking at the first four bytes in the display. Notice that the first two are D003. Two-byte values are usually stored with the least significant byte first so this is actually $03D0, the address you specified when the BSAVE command was issued. The next two bytes are 3000, so the length is $0030.

If you can read assembly language and want to see what is in a binary file, you can use ZAP's mini-disassembler. It's just like the one in the Apple monitor (in fact it uses the disassembler in the monitor!). Type the following:

```
    4
```

This tells ZAP to move the buffer cursor to the fourth byte of the sector buffer, past the address and length bytes to the first byte of the binary data. Now type:

```
    I
```

The hex/ASCII display has been replaced with a disassembly of the first 20 instructions. Since this memory was saved from $3D0, you are looking at the page 3 DOS jump vector table. To erase the instructions and see the hex/ASCII display again, press RETURN. Notice that the buffer cursor has been moved to point to the first byte following the last instruction disassembled. Thus, if you wanted to, you could "page" through all of the instructions in a sector by typing "I" repeatedly.

Another useful feature of ZAP is its trace table. Each time your screen is updated, ZAP makes an entry in an internal table of 32 entries called a trace table. Each entry contains your current position on the disk in terms of track/sector and buffer offset or RSA and buffer offset. To back up to the last place you have been you need only enter the following command:

```
    <
```

Enter the less-than symbol now, followed by RETURN. The buffer cursor backs up to its position prior to issuing the I command. If you type < again the cursor will be positioned where it was when you first OPENed the file, at $00. You can go forward in the table too. Type a > and press RETURN. The cursor advances to the start of the binary data again. If you press > again you will be back to your last position, following the instructions you disassembled. Pressing the > once more gives the error message:

```
        END OF TRACE
```

ZAP can do many things for you but it can't see into the future! The 'END OF TRACE' message will also appear if you back up too far. If you are interested, you might type:

```
        TRACE
```

You will see a formatted list of all the trace entries in the current trace table with an indicator next to your current position in the table. The trace table is particularly handy when you are scanning through a file for some string and you want to back up to a previously found occurrence again. Note that OPENing a file clears any existing trace table entries. This is to avoid ending up in a position that is outside the bounds of the file.

Now close the file and return to absolute track/sector mode by typing the command:

```
        CLOSE
```

The T$ and S$ reappear on the status line indicating that a file is no longer open. We will now digress from the tutorial briefly to discuss how commands are entered to ZAP.

Most ZAP commands take one of three forms:

COMMAND type	**example**
no operands	VTOC
one or two numeric operands	R11,0
a string operand	OPEN'BINARY FILE'

For example, the VTOC command has no operands (none are needed). The Read and WRITE commands can have operands, however. For example, if one numeric operand is given following the R command (and no file is open) it is taken to be a track number (and sector 0 is assumed). If two are given, they are the track and sector number to use. The OPEN command has a single string operand, the file name. String operands can be given either as characters enclosed within single quotes or as two digit hexadecimal values following an optional dollar sign. For example, the following two commands are identical:

```
        OPEN'HELLO'
             or
        OPEN$C8C5CCCCCF
```

In addition, you may use double quotes to specify ASCII characters where the most significant bit (MSB) is off (the OPEN command ignores the MSB in any case but it may make a difference on other commands).

Numeric operands may be given in hex, decimal or character. For example:

```
        R$11,12.
        R,F
```

Notice that decimal numbers must have a decimal point after them. Hex numbers may be preceeded by a dollar sign but this is not required. A numeric operand can also be in the form of an expression. Expressions consist of numeric values added or subtracted from one another:

```
        R$11+'C'-'A'+2.
        R,+1
```

The last example here points out the fact that most numeric operands can be relative as well as absolute. In this case you are saying "add one to the current sector number and read" or, in other words, "read the next sector". To back up one sector you would type:

```
        R,-1
```

ZAP provides a shorthand way for going to the next sector or backing up one sector. The N command goes to the **next** sector and the P command goes to the **previous** sector. These commands are identical to those shown above. You can even move forward several sectors:

```
        R,+99.
```

Another way of doing this is:

```
        N99.
```

Likewise, if a file were open you could move forward 5,000 bytes in the file by typing:

```
+5000.
```

You can experiment with expressions by using ZAP's built in calculator command. Just type a question mark followed by an expression. For example:

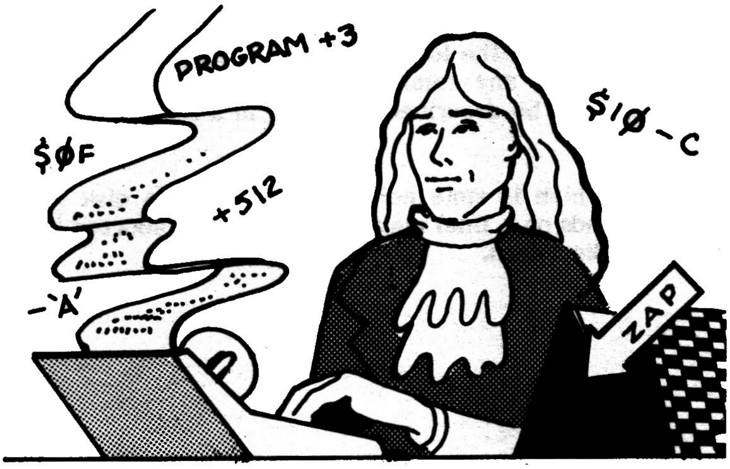
```
?512.-$FF+1
```

Enter the command above and press RETURN. The following message is displayed:

```
258. $000102
```

The results of the expression calculation are given in both their hexadecimal and decimal form. You'll find the ZAP calculator command is handy for computing offsets as well as converting hex to decimal and decimal to hex.

This ends the introductory tutorial. All of the ZAP commands will now be covered in detail. Additional tutorial-like uses for ZAP are provided in Chapter 6.



THE ZAP CALCULATOR COMMAND

ZAP COMMANDS

In general, ZAP commands consist of one or more characters of command name followed by zero, one, or two operands. No spaces may appear between the command name and its operands. For example:

```
WRITE11,0
```

is valid but:

```
WRITE 11,0
```

is not. This restriction is necessary since multiple ZAP commands may appear on one line, separated by blanks. For example:

```
N +3 :'HI THERE' UNLOCK WRITE LOCK
```

will advance to the next sector, move the buffer cursor forward 3 bytes, store the string 'HI THERE' into the buffer at the current position, unlock ZAP to allow writes to the diskette, write the modified sector image back to the diskette, and re-lock ZAP — all without any additional user intervention. Obviously, putting a blank between a command and its operands will cause ZAP to interpret the command as if no operands were given and to attempt to interpret the operands as a second command on the line!

If an error occurs during the execution of any command, an error message will appear on the left half of the command line. Execution of the command and any commands following it will be aborted. Error messages will be described in detail in a later section of this chapter.

Any operation which is time consuming (such as a disk search or multiple commands on a line) may be prematurely aborted by typing any key on the keyboard.

ZAP commands fall into several categories:

INPUT/OUTPUT COMMANDS
BUFFER MODIFY COMMANDS
COMPARISON COMMANDS
OPTION SWITCH COMMANDS
FILE COMMANDS
PRINTER COMMANDS
THE BUFFER COMMAND
MACRO COMMANDS
LABEL COMMANDS
TRACE COMMANDS
MISCELLANEOUS COMMANDS

Each of these categories will be covered now. The general format of each command will be shown first, followed by a description of its use. Wherever the string /EXP/ appears, you may replace it with any valid expression whose computed result will be used for the value of the operand. Do **not** type the slashes. Expressions are formed by combining one or more terms, such as hexadecimal numbers, decimal numbers followed by a decimal point, character strings (see below), or label variables (to be described later), using addition (+) or subtraction (-). Examples of valid expressions are:

15	(hex $15)
15.	(decimal 15)
'B'-157.+A	(ASCII for B less 157 decimal plus hex A)
PROGRAM+3E3	(Label variable plus offset in hex)
+ENTLEN	(Add label variable to current value)

/STR/ represents a string operand, either hex or characters. A hex string may be up to 16 bytes in length and may be preceded by an optional dollar sign. A character string may be up to 32 characters in length and must be surrounded in either single or double quotes. If single quotes are used, the ASCII representation of each character is assumed to have the most significant bit **on** (standard Apple practice). If double quotes are used, the MSB is assumed to be **off** (CP/M and PASCAL practice). Examples of valid strings are:

$3D0F56BE7F	(Hex string of 5 bytes)
5	(hex string of 1 byte)
'HI THERE'	(Character string of 8 characters, MSB on)
"CPM FILE"	(Character string of 8 characters, MSB off)

INPUT/OUTPUT COMMANDS

/EXP/

Move the buffer cursor to the offset given by /EXP/. /EXP/ may range in value from $00 to $FF (0 through 255.).

+/EXP/ ...or... -/EXP/

Move the buffer cursor to a new offset, computed by adding or subtracting the value of the expression, /EXP/, to or from the current buffer offset. If the result is prior to or after the current sector, ZAP will read the appropriate new sector first. For example, if the command +256. is issued, ZAP will read the next sector on the diskette (or in the file) and position the buffer cursor to the same offset as before. The command +257. would position the cursor in the next sector at one byte beyond its original position. Values for /EXP/ may be from -8388608. to +8388607.. If the newly computed sector is past the end of the current track, the track number will be modified as well. Likewise, wraparound will occur at the beginning and end of the diskette or file (if in the WRAP mode).

R/EXP1/,/EXP2/ ..or.. R/EXP1/ ..or.. R,/EXP2/ ..or.. R

NO FILE OPEN: Read the track and sector indicated by /EXP1/ and /EXP2/ respectively. If the sector number is omitted, as in the second format shown above, zero is assumed. If the track number is omitted, as in the third format, the current track number is assumed. If both are omitted, the current track/sector is reread. /EXP1/ and /EXP2/ may be either absolute or relative. If relative, wraparound will occur at the beginning/end of the diskette or file (if in WRAP mode).
FILE OPEN: If a file is open, /EXP1/ represents the record number and /EXP2/ represents the byte offset in that record. The record's position is computed by multiplying the record length specified by the RLEN command by the record number given, resulting in the absolute byte offset into the file.

WRITE/EXP1/,/EXP2/ ..or.. WRITE/EXP1/ ..or.. WRITE,/EXP2/ ..or.. WRITE

NO FILE OPEN: Write the contents of the current sector buffer out to the diskette at the track and sector indicated by /EXP1/ and /EXP2/ respectively. If the sector number is omitted, zero is assumed. If the track number is omitted, the current track is assumed. If both are omitted, as is usually done, the buffer is written to the current track/sector. As with the R command, /EXP1/ and /EXP2/ may be absolute or relative.
FILE OPEN: If a file is open, /EXP1/ and /EXP2/ indicate the record and byte offset. In this case, the data in the buffer is written to the sector containing this byte.

N/EXP/ ..or.. N

NO FILE OPEN: Move to the next sector. If /EXP/ is used, add the value of /EXP/ to the current sector number and read the new sector. If no /EXP/ is given, a value of +1 is assumed. /EXP/ may be any absolute value from -8388608. to +8388607.. Passing over a track boundary will result in moving to the next track. Wraparound will occur at the disk or file boundaries (if in WRAP mode).
FILE OPEN: If a file is open the N command reads the next **record** in the file.

P/EXP/ ..or.. P

NO FILE OPEN: Move to the previous sector. If /EXP/ is used, subtract the value of /EXP/ from the current sector number and read the new sector. If no /EXP/ is given, a value of 1 is assumed (back up one sector). /EXP/ may be any absolute value from -8388608. to +8388607.. Passing over a track boundary will result in moving to the previous track. Wraparound will occur at the disk or file boundaries (if in WRAP mode).
FILE OPEN: If a file is open the P command reads the previous **record** in the file.

%

Indirect read command. Used only in absolute mode (no open file). The % command looks in the sector buffer at the current offset for a two byte track/sector pair, then reads the indicated sector into the buffer. This command is handy when following a chain of track/sector pointers on a diskette, such as in the catalog or in a track/sector list.

BUFFER MODIFY COMMANDS

:/STR/ ..or.. :

Store command. The colon command replaces the contents of the buffer at the buffer cursor location with the string operand. The string may be hex or character. If no string is given, then the previous store string (if any) is used. The store string is the last operand of any buffer modify command (:,SET,&,O,X). This allows multiple stores of the same string without having to retype the string over and over again. The colon command is the only ZAP command which may follow another on the line without an intervening blank. This allows a construction similar to the Apple monitor (3E:00, for example)

SET/STR/ ..or.. SET

Multiple store command. The SET command will set the remainder of the buffer (from the buffer cursor on) to the specified string. If no string is given, the previous store string is used. For example, you can set the entire buffer to zero by moving the buffer cursor to offset 0 and issuing a SET0 command. If a string of more than 1 byte is given, it will be repeated in the buffer until there is not enough room left to store the full length of the string again.

&/STR/ ..or.. &

Logical AND command. The ampersand command will perform a logical AND function between the given string and the contents of the buffer at the buffer cursor location. If no string is given the last store string is used. For example, &7F will turn off the MSB of the byte at the current buffer cursor location.

O/STR/ ..or.. O

Logical OR command. The O command will perform a logical OR function between the given string and the contents of the buffer at the buffer cursor location. If no string is given the last store string is used. For example, O80 will turn on the MSB of the byte at the current buffer cursor location.

X/STR/ ..or.. X

Logical Exclusive OR command. The X command will perform a logical exclusive OR function between the given string and the contents of the buffer at the buffer cursor location. If no string is given the last store string is used. For example, X01 will make an odd byte even or an even byte odd.

COMPARISON COMMANDS

L/STR/ ..or.. L

Look command. The L command searches the buffer for the given string starting with the byte following the buffer cursor, and, if an occurrence of the string is found, the buffer cursor is left over the first byte of the matched string. If an occurrence cannot be found in the buffer, the next sector is read and also searched. This continues until a match is found or until the search wraps around, back to its starting location (if in WRAP mode). Searching can be done in absolute mode (no file open) when the entire disk will be searched, or in file mode (file open) when just the file is searched. Matches can occur even where the string crosses a sector boundary. If no string operand is given, a previous search resumes using the current comparison string (from the last L or V command). Thus, you may start a search, and, after finding an occurrence, look for further occurrences by typing L. The search may be interrupted at any time by typing any key.

V/STR/ ..or.. V

Verify command. The V command verifies that the string under the buffer cursor matches the given string. If it does not, an error message is displayed. If no string is given, the current comparison string is used. The V command can also be used to delimit the bounds of a search. Position to the last sector to be searched and type a V command with the search string to be used. Position to the first sector to be searched and type L without entering the search string. The look command will use the search string and search boundary established by the previous V command.

COMPARE/EXP/

The COMPARE command compares the contents of the current buffer to another buffer, from the buffer cursor location to the end of the buffer. /EXP/ is the buffer number of the buffer to be compared to the current one. Using the # command (described later) you can read two sectors into two different numbered buffers (one into 0 and one into 1, for example). You can then type COMPARE1 while you are displaying buffer 0 and this command will compare the two images, byte by byte. If they do not match, the buffer cursor is left over the first byte which differs and an error message is displayed.

OPTION SWITCH COMMANDS

Most option switch commands act like binary switches. This means that the commands toggle an internal ZAP "switch" which can be in either of two positions. These switches have an initial default setting, but you may change them using option switch commands. Option switches control ZAP's operation in numerous ways, including display of ASCII, the LOCK mode, etc.

LC

Sets the ASCII translation on the right hand side of the hex/ASCII display so that lower case characters will be printed "as is". This mode is useful for users who have a lower case modification on their Apple or when DUMPing sectors to a printer which supports upper and lower case. The default setting for this switch is with LC off. LC only has meaning when the ASCII switch is in effect.

UC

The opposite of LC. UC instructs ZAP to translate lower case characters to upper case for display on the right half of the screen. UC mode is the default. UC only has meaning when the ASCII switch is in effect.

IMAGE

Sets the ASCII translation on the right hand side of the screen so that minimal translation is done. Inverse and flashing characters appear as is and only control characters which would adversely affect the display are translated out. While IMAGE mode is in effect, UC and LC modes are ignored. The default mode is ASCII, not IMAGE.

ASCII

The opposite of IMAGE mode. UC or LC translations are done and non-printing control characters are translated to periods. Flashing and inverse video characters are translated to normal. ASCII mode is the default.

LOCK

Sets ZAP into LOCK mode. While in LOCK mode ZAP will no allow you to write to the diskette. LOCK mode is the initia default.

UNLOCK

Resets LOCK mode allowing ZAP to write to the diskette.

DOS16

Informs ZAP that the diskette is a 16 sector DOS diskette. ZAP uses this information when selecting the proper sector skewing table, setting up its RWTS (Read/Write Track/Sector) package for 13 or 16 sector I/O, and for determining the catalog format for the OPEN file command. After setting DOS16 mode, ZAP erases the trace table, empties all of the buffers, and reads the first catalog sector into the current buffer.

DOS13

Informs ZAP that the diskette is a 13 sector DOS diskette. After setting DOS13 mode, ZAP erases the trace table, empties all of the buffers, and reads the first catalog sector into the current buffer.

CPM

Informs ZAP that the diskette is a 16 sector CP/M diskette. Processing is otherwise similar to that of DOS16.

PASCAL

Informs ZAP that the diskette is a 16 sector PASCAL diskette. Processing is otherwise similar to that of DOS16.

WRAP

Sets ZAP into WRAP mode. This is the default initial state. While in WRAP mode, ZAP will allow wraparound when an attempt is made to read/write beyond the end of a diskette or file or before the beginning.

NOWRAP

Turns off WRAP mode. When an attempt is made to read/write beyond the end of a diskette or open file (or before the beginning of them) an error message is displayed. This command is useful when it is necessary to scan from the current location to the end of a file or disk, without wrapping back to the beginning again.

FILE COMMANDS

As described in the tutorial that began this chapter, ZAP normally operates in an absolute mode. While in this mode, all references are at the track/sector level. If the OPEN file command is issued, however, ZAP switches to a file relative mode. While in file mode, all references to sectors are file relative. This means that the N (next) command, for example, will read the next **logical** sector of the file, even if that sector is not the next physical sector on the track. While in file mode, no absolute track/sector references may be made and only sectors contained in the file may be accessed.

Having a file open modifies the operation of the R (read) and WRITE commands as well. References are at the record/byte offset level, rather than track/sector. The record length is initially set to 256., equating record numbers to relative sectors, but may be changed using the RLEN command.

OPEN/STR/ ..or.. OPEN

Opens the file named by /STR/. If no string is given, the last file opened is reopened. If a file is already open, it is closed, and the new one is opened. ZAP searches the directory (based upon the type of diskette in use) for the file name string. The file name string may be given in hex or character (the MSB is ignored) and may be specified in an abbreviated form, much like that of the Apple utility, FID. If the last character of the name is an equal sign (=), ZAP will select the first file name in the directory which starts with the string. Once a file is found in the directory, ZAP will build a map of the file's sectors in memory. This map can contain up to 512 relative sectors. In most cases this is sufficient even if the file spans the entire diskette. If the file is an extremely large, very sparse, random type text file, however, only the first 512 sectors may be accessed. After building the sector map, ZAP reads the first data sector into the buffer. The trace table is cleared and the buffers are marked empty. Note that CPM and PASCAL file names which do not have a suffix (such as .CODE or .SYS etc.) must be specified with a trailing period. For example, a file named GORF should be given as OPEN"GORF." or a file not found message will occur.

RLEN/EXP/

Sets the record length for use with the R (read) and WRITE commands. The OPEN command sets the initial record length to 256. The record length is multiplied by the relative record number given on the R or WRITE command to compute the relative byte offset into the file. The record length may range in value from 1 byte to the length of the file in bytes. Note that changing the record length will affect the N and P commands also.

CLOSE

Causes ZAP to exit file mode and return to absolute mode. The RSA (Relative Sector Address) is replaced on the status line by the Track/Sector of the current buffer.

WHERE

When in file mode, the WHERE command works identically to the STATUS command. If no file is open (absolute mode), the WHERE command will search each file in the diskette's directory to determine whether it contains the current track/sector. If such a file is found, it is opened, the sector is reread, and a STATUS command is invoked to display the location of the sector in the file. The WHERE command is particularly useful in identifying the file containing a sector with an I/O error.

PRINTER COMMANDS

PR#/EXP/

Sets the slot number to be used with an optional printer. The default is 1. If a printer controller card is plugged into this slot, you may issue commands to dump sectors to the printer or produce a LOG listing (audit trail). The printer slot in use is displayed on the status line (the top line of the display).

PRINT

Copies the entire screen image to your printer. This command may be issued to print any ZAP screen image, including the macro table display, the label table display, and the help screens. If your printer echos what it is printing on the screen this command may not work properly and the DUMP command below must be used to dump out sector contents. Note that some printer interface cards, such as the Apple Serial Interface Card, allow you to inhibit print echoing.

DUMP/EXP/ ..or.. DUMP

Dumps the current sector buffer in hex and ASCII onto the printer. A status line is also printed. The ASCII translation used is affected by the ASCII/IMAGE/UC/LC switches. If an expression is given, it represents the number of consecutive sectors (or records) to be dumped. If no expression is given, only the current sector is dumped.

IDUMP

Disassembles and prints the 6502 instruction equivalents for the data in the sector buffer, starting at the current buffer cursor location and proceeding to the end of the buffer. A status line is also printed.

NOTE/LINE/

Prints a comment on the printer. The remainder of the line, following the command NOTE, is printed on the printer as is (no other ZAP commands may be issued on the same line following a NOTE command). NOTE is useful to label a listing. For example: NOTE THIS IS A PATCH TO DOS3.3 TO PREVENT ITS DESTROYING THE LANGUAGE CARD.

LOG

Turns on LOGging mode. While in effect, the printer slot number on the status line is displayed in inverse video. Each time ZAP reads or writes a sector a status line is printed on the printer. Also, any time a store (colon) command is invoked, a status line and the "before" and "after" hex strings are printed. In this way, ZAP keeps a complete record, or "audit trail", of your activities, allowing you to determine later what was done, and, if necessary, "back out" bad changes. Note that LOG mode produces a lot of output should you use the Look command, OPEN, or WHERE (since many sector reads are done). For this reason, it is recommended that LOGging be turned on only when absolutely necessary.

NOLOG

Turns off LOGging mode. This is initially the default mode and is indicated by a normal video representation of the printer slot on the status line.

THE BUFFER COMMAND

ZAP maintains sixteen (16) 256 byte areas, called buffers, into which you may read sector images. More than one buffer is provided to allow you to simultaneously manipulate multiple sectors without having to write any of them back to disk until all changes are made. Likewise, multiple buffers are handy when writing macros to compare or copy files or complete diskettes. The current buffer number (in hexadecimal) is displayed on the status line next to the # sign. Initially buffer zero (#0) is displayed. Each buffer has associated with it a track and sector number for the sector image it holds (if a file is open this is also true, although in this case it is in terms of a relative sector address (RSA)). This "tag" information is set whenever an INPUT/OUTPUT COMMAND is issued for that buffer. If no I/O command has ever been issued or if the type of diskette (DOS16, etc.) has changed or if a new file is OPENed, these tags are erased (since they are no longer meaningful) and the buffer is considered "empty". An empty buffer is denoted on the status line with question marks in the track/sector (or RSA) fields but the data remains intact to allow you to copy or compare data across different files or diskette formats.

#/EXP/

The # command selects a buffer (from $0 through $F) to be displayed and manipulated. The expression may be absolute or relative (for example, #+1 is valid as long as you do not exceed the valid buffer number range).

MACRO COMMANDS

One of ZAP's most powerful features is the ability to write your own macros. A macro is, functionally, a string of ZAP commands, associated with a short name. Whenever you type the name, it is automatically replaced on the command line with the string of commands it represents. In this way, using a combination of ZAP commands, you can define your own commands. Macros are also handy to provide a shorthand way of performing multiple tasks. In Chapter 6 there are examples that use macros to copy and compare files.

Macro definitions are preceded by an open parenthesis and end with a close parenthesis. An example macro definition might be:

```
(ZOT UNLOCK WRITE LOCK)
```

Here, the first word in the definition is the name of the macro itself, ZOT. Each time the word ZOT appears on a command line it will be replaced by the string "UNLOCK WRITE LOCK" and these commands will then be processed. In this case, the ZOT command just defined will defeat the lock mode protection by unlocking, writing, and relocking — a quick and dirty way of forcing a write command to work. Incidentally, a macro definition may contain the name of another macro (nested macros) if you desire. For example, a new macro could be defined as follows:

```
(ZIT ZOT NOTE FORCED WRITE)
```

In this case, the ZIT macro invokes the ZOT macro and then prints a comment on the printer, "FORCED WRITE". Be careful, however. If a macro invokes itself, either directly or indirectly, recursion will occur and you will fill up the command line and an error message will be displayed. The final size of the command string, after all macro string replacements have been made, may not exceed 256 characters.

To some extent, a macro can have operands. Consider the following macro:

```
(Z WRITE)
```

Whenever the Z macro is invoked, a WRITE command is performed. If you entered a command of the form:

```
Z11,0
```

the line would look like this after macro replacement:

```
WRITE11,0
```

Since the last command in the macro's string can take operands, then the macro can take operands as well.

ZAP maintains a 256 byte macro table. Using the macro definition command you may store several macros in this table. At most, 256 characters may be stored. The table might contain one huge macro or several smaller ones. You may save the entire table by using the MSWAP command, described later, or you may print it using the MACROS and PRINT commands. If you try to define a new macro but there is not enough room left in the table you will see a "TABLE FULL" error message. In this case, you will have to delete an old macro that you (hopefully) no longer need to make room for the new macro and retry the definition.

ZAP provides you with several built-in macros. These macros are predefined for you when you first enter ZAP. You may use them or delete them at your option. They have been chosen for their general utility as well as for being good examples of macros. They are:

MREAD - If you are currently viewing a sector in buffer zero, MREAD will read it and the next 15 sequential sectors into buffers 0 through 16, allowing you to modify multiple sectors (one track's worth) at a time.

MWRITE - Writes the contents of all of the buffers back to the diskette.

SAVEM - Saves the contents of the macro table by writing it to track $02, sector $0F (an unused sector on diskettes containing DOS 3.3 on tracks 0, 1, and 2) on the current diskette.

LOADM - Reloads the macro table from track $02, sector $0F of the current diskette.

REOPEN - If a file was previously open and you are still positioned to a sector belonging to that file, REOPEN re-opens the file and positions to that sector.

LAST - Positions to the last byte of an open file or the last byte of the diskette (WRAP must be allowed for LAST to work properly).

QUIT - An alias for the END command. Many users are more used to QUIT to exit a program then END.

The commands associated with macros are:

(/NAME/ /TEXT/)

Macro definition. The entire definition must be enclosed in parentheses. It may not contain another macro's definition. The first word is used as the name of the macro itself. The name may be of any length and may contain any characters or numbers. If a macro by this name already exists, the old macro is replaced with the new definition. Care should be used in choosing macro names to avoid conflict with existing ZAP commands, label variables or other macros. ZAP gives precedence to macros over its own commands, so, if you really wanted to, you could redefine a ZAP command to do something else. For example, (HELP NOTE THERE IS NO HELP FOR YOU). In this example, the true ZAP HELP command cannot be accessed until this macro is deleted. For this reason, (WRITE UNLOCK WRITE LOCK) would not work, since, in redefining the WRITE command, you have denied yourself further access to the original. The /TEXT/ is separated from the name of the macro by one space and may be of any form as long as it does not contain a closing parenthesis character. Another warning involves the way ZAP scans for commands. If you were to define a macro called "GO" and later define a macro called "GORF" one might expect that invoking the GORF macro would actually produce the GO macro followed by the operand "RF". ZAP will prevent this kind of thing from happening by deleting GO when you define GORF. It is still possible to run into trouble, however, if you define a macro whose name is a shortened version of a label variable or ZAP command. For example, WR is not a good name for a macro since it would prevent you from getting at the ZAP WRITE command.

//NAME/

The / command deletes the macro whose name is /NAME/ from the macro table.

MACROS

Displays all currently defined macros on the screen in place of the hex/ASCII dump. Press return to redisplay the hex/ASCII again.

MSWAP

Exchanges the contents of the current sector buffer with those of the macro table. The sector buffer must contain a valid macro table, previously created by the MSWAP command, or it must be precleared to zeros (use the command SET0 — this creates a macro table image which contains no macros). The table data should not be modified in any way while it is in the sector buffer as this will result in a "BAD DATA FORMAT" error message. Thus, to save the current contents of the macro table you could use the following commands:

```
SET0 MSWAP UNLOCK WRITE2,F MSWAP
```

This would preclear the buffer to zero, swap this "empty" macro table with the one you want to save, write the table to be saved to the diskette, and then swap it back into the macro table. To reload the macro table you would use the following commands:

```
R2,F MSWAP
```

The MSWAP command can be used by the sneaky macro programmer to extend the allowable space for macros beyond 256 bytes. This can be done by storing one or more additional macro tables in otherwise unused ZAP sector buffers and then MSWAPping them in as needed under the control of another macro.

LABEL COMMANDS

ZAP allows the definition of up to ten label variables. A label variable is much like a variable in BASIC. It consists of a name (from one to eight characters in length) with which ZAP associates a track number, sector number, and buffer byte offset (when a file is open, label variables represent an RSA and buffer byte offset). The usual use of label variables is to remember your place in a file or somewhere on the diskette. For example, imagine that you have, after much searching around, located the HELLO file name in DOS. You want to associate a "name tag" with this position on the diskette so you define a label as follows:

```
=HELLO
```

Later, after positioning to other sectors on the disk, you want to return to the HELLO file name's sector again so you type:

```
        HELLO
```

ZAP looks up the word HELLO in its macro table first and doesn't find a macro by that name. It then searches its label table and finds a definition for HELLO. The sector containing the HELLO file name is read and the buffer cursor is left over the first byte of the file name.

Labels may be followed by expressions. For example, if you have defined a label to point to a subroutine in a binary file which contains an assembly language program you are debugging, you can position to an offset in that subroutine. For example:

```
        SUBRTN+3E3
```

ZAP will compute the proper sector to be read and read it, leaving the buffer cursor at the appropriate offset.

Label variables can also be used as terms in an expression. When labels are employed in this way, the track and sector part of the label is ignored and only the buffer offset value (from $00 to $FF) is used. The reason for doing this can be made clear by the following example:

```
        35.
        =EL
        11.
        +EL
        +EL
        +EL
        etc.
```

Suppose you were looking at a DOS catalog sector. Each file's entry is 35 bytes long. By positioning to buffer byte 35 decimal (which has no special meaning, apart from being at the right offset) and assigning a label, EL (for entry length), we have a convenient way of moving from catalog entry to catalog entry. After positioning to byte 11 decimal, it is a simple matter to instruct ZAP to position +EL bytes forward each time we want to see where the next file's entry begins. To sum up, if a label appears first in a command string, its track and sector values are used, but if it appears later in an expression, only its buffer byte offset is used.

The same caveat exists for label variables as for macros. Try to avoid picking names which might conflict (even partially) with those of ZAP commands. Remember that your macro names take precedence over your label names, and your label names take precedence over ZAP command names.

One label, the * label, is predefined for you by ZAP. The * (asterisk) label always contains the position following the last read or write operation. * is very useful when reading into an "empty" buffer, since there is no other way to make a relative reference (R+3 or the N command are not valid in an empty buffer).

Commands connected with label variables are:

LABELS

Display all currently defined labels on the screen. To return to the hex/ASCII display, press RETURN.

=/NAME/

Define a new label (or redefine an old one) with the current position. Up to 10 labels (including *) may be defined. The name must be one to eight characters or numerals in length.

//NAME/

Delete a label from the table to make room for new ones. Note that this is the same as the delete macro command. You may not delete the * label.

/NAME/

Position to the track/sector/offset associated with the label whose name is /NAME/.

/NAME/+/EXP/ ..or.. /NAME/-/EXP/

Compute a location which is the sum (or difference) of the location associated with the label variable whose name is /NAME/ and the expression and read that sector into the buffer.

LSWAP

Exchanges the contents of the label table (not including the * label) with those of the current sector buffer. This command works almost identically to the MSWAP command, described in the section on macros. The sector buffer must contain a valid label table image, previously LSWAPped, or it must be set entirely to zeros. The label table may be saved in exactly the same way as is the macro table. (A DOS 3.3 diskette containing a boot image of DOS on tracks 0 to 2 does not use sectors 5 through F of track 2. These are safe places to store label and macro tables.) Note that it is possible using LSWAP for labels to become defined which are not within the bounds of a previously OPENed file or which describe sectors which do not exist on a 13 sector diskette. If these labels are referenced, error messages will result.

TRACE COMMANDS

As mentioned in the tutorial that began this chapter, ZAP keeps a 32 entry trace table that records your movements within a sector buffer or over the diskette or open file. This allows you to back up to a previous position and move forward again to your last position. Each time your display is updated, if the buffer cursor has moved or if a new sector has been read, an entry is made in the trace table. If the trace table is full, the oldest entry is removed to make room for the newest one.

TRACE

This command displays all of the trace entries and indicates which is current. Press RETURN to recover the hex/ASCII display.

<

Back up one trace table entry. If no more previous entries exist an error message is printed.

>

Move forward one trace table entry. If no more future entries exist, an error message is printed.

AT/EXP/ ..or.. AT

The AT command is similar to the /EXP/ command listed under INPUT/OUTPUT COMMANDS and the /NAME/ or /NAME/+/EXP/ commands listed under LABEL COMMANDS except that it will not actually read anything. Only the buffer tag information for the current buffer is changed, and it is changed to the value of /EXP/. The buffer tag is the track and sector (or relative record) information associated with the buffer.

The AT command is useful if you want to do a WRITE operation using an expression or label variable. For example:

```
ATPROGRAM+3E3
WRITE
```

will set up the tag information to point to the sector which is +$3E3 bytes from the label "PROGRAM" and then write the contents of the buffer to that sector.

If no expression is given with the AT command, the current buffer is marked empty (the tag information is set to question marks). This is useful sometimes when you want to force ZAP to read a sector in, even if the track/sector associated with the buffer matches the one you are reading. For example, you might perform the following commands:

```
#+1 AT PROGRAM 3E:01 WRITE
```

In this example, if the AT command did not precede the label expression, PROGRAM, it is possible that the new buffer was already pointing there but that the data in it was not useful for some reason (you had switched disks or had changed the buffer data in some way). Putting the AT command first forces ZAP to read a sector from diskette when it sees the PROGRAM expression.

@ is an alias for the AT command.

S/EXP1/,/EXP2/ ..or.. S/EXP1/ ..or.. S,/EXP2/

The S command sets the slot and/or drive for subsequent disk accesses. The slot and drive number are initially set to those used to boot the Bag of Tricks diskette. If you want to change them you may use this command. The first expression, /EXP1/, is the slot number and may be relative or absolute, ranging in value from 1 to 7 (there must be a disk controller in the slot specified, however). /EXP2/ is the drive number, 1 or 2, and may be relative or absolute (for example, S,-1 is valid if the current drive is 2). If /EXP1/ is omitted, the current slot remains unchanged. If /EXP2/ is omitted, drive 1 is assumed. You may not change slots or drives with a file open as this would confuse ZAP.

?/EXP/

The calculator command. The value of the expression is printed on the command line in hexadecimal and decimal. The expression may be any valid ZAP expression, including those containing label variables. ZAP expressions may contain addition and subtraction operations.

I

Instructions command. ZAP disassembles as many 6502 instructions as will fit on the screen from the current buffer cursor location forward. If the end of the buffer is reached, disassembly stops and the buffer cursor is positioned to $00. The buffer cursor is normally left so that a subsequent I command will pick up where a previous one left off. Pressing RETURN will return to the hex/ASCII display.

LOOP/EXP1/,/EXP2/ ..or.. LOOP/EXP1/ ..or.. LOOP,/EXP2/ ..or.. LOOP

The LOOP command can be used to cause ZAP to repeatedly execute all, or part of the command line. Only one loop may be in effect at a time (nested loops are not permitted and will result in unpredictable behavior). In its simplest form, LOOP with no operands will repeat the entire command line that precedes it, from the first character up to the LOOP command, ad infinitum (forever and a day), or until an error message is displayed (you ran off the end of the buffer, for example) or until you press any key on the keyboard. This form is the easiest to use:

```
        UNLOCK NOWRAP SET0 WRITE N LOOP
```

will zero out the entire diskette (from the cursor on) or an entire file if a file is open. The command will stop when the "NUMBER TOO BIG" error occurs as ZAP attempts to read off the end of the diskette (or file) and wrapping is not permitted.

If /EXP1/ is given, it is used to count the number of times the loop is to be performed. It may be any positive absolute number from 1 to +8388607. For example:

```
        #+1 AT *+256. LOOP15.
```

will fill all 16 buffers with the next 16 sectors (assuming buffer zero has already been read).

If /EXP2/ is given, it is used to identify the command at which to start looping when you don't want to start looping with the first command on the line. The value of /EXP2/ identifies the character in the command line where the looping is to start. The first character is numbered 0 and blanks are counted. Consider the following example:

```
        #0 * #+1 AT *+256. LOOP15.,5
```

In this example, character 5 on the command line is the place to which ZAP is to loop back. This would be the first character of the #+1 command. A relative expression could also be used:

```
        #0 * #+1 AT *+256. LOOP15.,-18.
```

This line is identical to the command line given earlier except that a relative offset from the first operand character of the LOOP command is given (#+1 is 18 decimal characters back from the first character of 15.,-18.). This form is preferable if you are writing a macro since you do not necessarily know where the macro will appear on the command line and can not be sure of the absolute line location of anything within your macro. Be careful, however, if you have any nested macro calls between the LOOP and its starting point. The macro replacement will change the number of characters between the start point and the LOOP command.

HELP/EXP/ ..or.. HELP

The HELP command provides you with a quick, on-line reference to all of ZAP's commands. There are five screens of help information available, which can be called up by giving a screen number as an expression (1 through 5). If no expression is given, the first help screen (screen 1) is displayed. Press the RETURN key to return to the hex/ASCII display.

VTOC

If the diskette type is DOS16 or DOS13, the VTOC command will read track $11, sector $00, the DOS Volume Table of Contents.

CAT

The first sector of the catalog (or file directory) is read. The actual location of this sector depends upon the type of diskette in use (DOS, CPM, or PASCAL).

STATUS

The STATUS command displays on the video screen several pieces of information about current ZAP status. The information presented includes:

- The name of the last OPENed file (or the current open file name)
- If a file is open...
  - The current position in record and byte offset form
  - The current position in track and sector form
  - The record length in use
  - The size of the file in records
  - The size of the file in bytes
- The status of ZAP option switches...
  - Buffer changed or unchanged
  - Lower case (LC) or upper case (UC) translation
  - Image or ASCII mode
  - Locked or Unlocked for WRITEs
  - Logging or not logging
  - Wrap allowed or not allowed

Number of sectors per track
Number of tracks per diskette
The last Look/Verify string, if any
The Look starting position (track/sector/offset)
The last store string, if any

Press RETURN to redisplay the hex/ASCII dump.

END

The END command exits ZAP and returns to the Bag of Tricks main menu screen.

ZAP ERROR MESSAGES

WRITE PROTECTED

Either ZAP is in LOCK mode and you have attempted to write to the diskette or your diskette is really write protected. If you still want to write, verify that ZAP is not locked by entering the UNLOCK command and check your diskette's write protect notch to make sure it is not covered. Then enter the WRITE command again.

DRIVE ERROR

Something is seriously wrong with the sector ZAP is trying to read or write. Possible problems might be:

The diskette door is open or there is no diskette in the drive
The sector has been damaged in some way
You are trying to read/write a 13 sector diskette but you told
ZAP that it was a 16 sector diskette.

If you were trying to read, try the command again. If it still doesn't work, try reading a neighboring sector or one on a neighboring track first and then try the read again. If the data in the sector is not important, set the buffer to zeros and WRITE it. If you get the same error, you may need to use the INIT utility to reformat the sector.

READ ERROR

See DRIVE ERROR.

NO VTOC

The VTOC command is not valid for CPM or PASCAL type diskettes.

SYNTAX ERROR

The command or its operands are invalid or improper in format. Remember that operands must follow the command with no intervening blanks. Watch out for macros or label variables defined by you which might conflict with each other or with ZAP commands.

WARNING: BUFFER CHANGED

ZAP is warning you that, since you read the sector into the buffer, you have made changes to it. If you don't care and want to throw these changes away, retype the command you just entered. Otherwise, WRITE the buffer first.

NOT FOUND

The file you attempted to OPEN was not found in the diskette's directory (catalog) or contains no sectors. This message also appears if you attempt to delete a non-existent macro or variable label. Use the MACROS or LABELS command to find out what your macros and labels are named.

NO FILE IS OPEN

Several commands are not valid unless a file is open (for example RLEN, CLOSE, etc.) Use the STATUS command to find out what is going on.

LINE OVERFLOWED

During macro replacement the total length of the command line exceeded 256 characters. No further processing can be attempted. Check your macros to make sure none of them invokes itself, thereby infinitely filling up the command line.

SCAN ENDED

A Look command which was originally invoked at the current disk location has returned to that location after finding no (further) occurrences of the string.

NOT IN FILE

You have attempted to read/write a sector which is not contained by the currently OPEN file (this is hard to do but not impossible). If you must access the sector, CLOSE the file first.

EMPTY BUFFER

Some commands are not valid for buffers which are marked "empty". Notably, you may not make any references which require a current track, sector or RSA value. The commands, R+3 or N or P would be examples of this. If you want to read the sector following the last sector read, use *+256. instead.

NUMBER TOO BIG

The value of an operand expression exceeds the valid range for that operand. This could happen, for instance, if you tried to read track 53 (only tracks from 0 to 34 decimal are valid). If your intent was to cause wraparound, give the number as an expression, relative to the current location (+53.).

NUMBER TOO SMALL

The value of an operand expression is too small. Most likely, you have given a negative number or zero and this is not valid. Drive numbers, for example, must be 1 or 2, not 0 or -197.

TABLE FULL

Either a macro definition or a label definition has been attempted and ZAP's corresponding table can not accept any more new entries. Delete an old entry to make room for your new one and try again.

MATCH

This is not really an error but this message will stop execution of multiple commands. A Look, Verify, or COMPARE command was issued and the strings matched. This message will be suppressed if any other commands follow the Look or COMPARE command on the command line.

COMMAND HALTED

This indicates that ZAP was processing commands but you interrupted it by pressing a key on the keyboard. It is not clear how far ZAP got with your command(s) before it was stopped.

END OF TRACE

You have attempted to move forward in the trace table but there are no more entries or you have attempted to back up and their are no more earlier entries. Use the TRACE command to view the entire trace table.

BEYOND BUFFER

You attempted a store operation with a string which would have run off the end of the buffer or you have attempted to position the buffer cursor outside the $00 to $FF range.

MISSING OPERAND

A command was entered which requires operands (such as S, set slot/drive) but no operands were given. The command does not provide a default value.

NOT A DISK

An attempt was made to set the disk slot number (with the S command) to a slot which does not contain a valid disk controller card. If ZAP had not checked first, it could have hung when you next tried to read or write. Verify that the slot you specified is a disk drive. ZAP checks by comparing the third and fifth bytes of the bootstrap ROM with Apple's standard for a disk controller card. This technique is identical to that employed by PASCAL.

CLOSE FILE FIRST

Some commands are not meaningful when a file is open because they would cause ZAP to read sectors which are not within the file. Examples of these are VTOC and CAT. Likewise, switching drives with a file open would cause ZAP to assume the file existed in the same location on the other drive, an unlikely expectation at best. For this reason, you must CLOSE any open file before switching drives.

BAD TRK/SEC PTR

Although ZAP attempts to validity check all track/sector values you give it and issue appropriate messages (such as NUMBER TOO BIG, etc.) it is possible, during an OPEN for example, that a bad track/sector pair will be used to perform a read operation. In this case the message appears. It usually means that you have attempted to OPEN a file on a CPM diskette, for instance, after telling ZAP that it is a DOS diskette. Another possibility is that some directory or track/sector list pointer is invalid. You should not see this message very often (hopefully).

DOES NOT MATCH

The result of a Verify or COMPARE command is that the strings or buffers do not match. In the case of COMPARE, the buffer cursor indicates the location of the mismatch.

OFF END/NO WRAP

An attempt has been made to move past the bounds of the diskette or an open file (by asking for a sector which is prior to the beginning of the disk/file or past the end) and the NOWRAP option has been set. The offending command has been aborted.

BAD DATA FORMAT

An MSWAP or LSWAP command has determined that the data in the sector buffer is not a valid macro or label table. These commands store a checksum byte in the data they produce to insure the integrity of their table formats. Do not change the data in any way while it is in the sector buffer. If you have not changed the data, then it was probably damaged when it was written to or read back from a diskette.

SUMMARY OF ZAP COMMANDS

INPUT/OUTPUT COMMANDS

/EXP/	SET BUFFER CURSOR
+/EXP/	MOVE FORWARD IN BUFFER, DISK, OR FILE
-/EXP/	MOVE BACKWARD IN BUFFER, DISK, OR FILE
R/TRK/,/SEC/	READ TRACK, SECTOR (FILE CLOSED)
R/REC/,/BYT/	READ RECORD, BYTE (FILE OPEN)
WRITE/TRK/,/SEC/	WRITE TRACK, SECTOR (FILE CLOSED)
WRITE/REC/,/SEC/	WRITE RECORD, BYTE (FILE OPEN)
N/EXP/	NEXT SECTOR (PLUS /EXP/ SECTORS)
P/EXP/	PREVIOUS SECTOR (MINUS/EXP/SECTORS)
%	INDIRECT READ TRACK, SECTOR

BUFFER MODIFY COMMANDS

:/STR/	STORE STRING INTO BUFFER
SET/STR/	MULTIPLE STORE
&/STR/	LOGICAL AND OPERATION
O/STR/	LOGICAL OR OPERATION
X/STR/	LOGICAL EXCLUSIVE OR OPERATION

COMPARISON COMMANDS

L/STR/	LOOK FOR STRING
V/STR/	VERIFY STRING MATCHES BUFFER
COMPARE/BUFFER/	COMPARE BUFFERS

OPTION SWITCH COMMANDS

LC	DISPLAY LOWER CASE AS IS
UC	TRANSLATE LOWER CASE TO UPPER CASE
IMAGE	PRINT CHARACTERS IN IMAGE FORM
ASCII	STANDARD ASCII TRANSLATION
LOCK	PREVENT WRITE OPERATIONS
UNLOCK	ALLOW WRITE OPERATIONS
DOS16	USE DOS 3.3 SKEW TABLE (16 SECTOR)
DOS13	USE DOS 3.2 OR 3.1 SKEW TABLE (13 SECTOR)
CPM	USE CPM SKEW TABLE (16 SECTOR)
PASCAL	USE PASCAL SKEW TABLE (16 SECTOR)
WRAP	ALLOW DISK OR FILE WRAPAROUND
NOWRAP	PREVENT DISK OR FILE WRAPAROUND

FILE COMMANDS

OPEN/STR/	OPEN A FILE
RLEN/EXP/	SET RECORD LENGTH
CLOSE	CLOSE FILE
WHERE	OPEN FILE CONTAINING SECTOR

PRINTER COMMANDS

PR#/EXP/	SET PRINTER SLOT NUMBER
PRINT	COPY SCREEN TO PRINTER
DUMP/EXP/	DUMP SECTOR(S) TO PRINTER
IDUMP	DUMP INSTRUCTIONS TO PRINTER
NOTE/LINE/	PRINT COMMENT LINE
LOG	LOG ALL CHANGES
NOLOG	STOP LOGGING CHANGES

BUFFER COMMAND

#/EXP/	SELECT BUFFER

MACRO COMMANDS

(/NAME/ /TEXT/)	DEFINE MACRO
/NAME/	INVOKE MACRO
//NAME/	DELETE MACRO
MACROS	LIST ALL MACROS
MSWAP	SWAP MACRO TABLE WITH BUFFER

LABEL COMMANDS

LABELS	DISPLAY ALL LABELS
=/NAME/	DEFINE LABEL
//NAME/	DELETE LABEL
/NAME/	POSITION TO LABEL
/NAME/+/EXP/	POSITION TO LABEL PLUS EXPRESSION
LSWAP	SWAP LABEL TABLE WITH BUFFER

TRACE COMMANDS

TRACE	DISPLAY TRACE TABLE
<	BACK UP IN TRACE
>	ADVANCE IN TRACE

MISCELLANEOUS COMMANDS

AT/EXP/	POSITION BUT DO NOT READ
AT	MARK BUFFER EMPTY
S/SLOT/,/DRIVE/	SET DISK SLOT,DRIVE
?/EXP/	CALCULATOR
I	DISASSEMBLE TO SCREEN
LOOP/CNT/,/LOC/	REPEAT LINE
HELP/EXP/	SHOW HELP SCREEN
VTOC	READ DOS VTOC
CAT	READ FIRST CATALOG SECTOR
STATUS	SHOW ZAP STATUS VARIABLES
END	EXIT ZAP

ALPHABETICAL LISTING OF ZAP COMMANDS

The following list presents the ZAP commands in alphabetical order followed by the page number where a complete description of each command may be found.

#	4-27
%	4-18
&	4-19
( )	4-30
+/EXP/	4-16
/	4-30,4-33
/EXP/	4-16
:	4-18
=	4-33
?	4-36
@	4-35
ASCII	4-21
AT	4-35
CAT	4-38
CLOSE	4-24
COMPARE	4-20
CPM	4-22
DOS13	4-22
DOS16	4-22
DUMP	4-25
END	4-39
HELP	4-38
I	4-36
IDUMP	4-26
IMAGE	4-21
L	4-20
LC	4-21
LOCK	4-22
LOG	4-26
LOOP	4-36
LSWAP	4-34
MACROS	4-30
MSWAP	4-31
N	4-17
NOLOG	4-26
NOTE	4-26
NOWRAP	4-23
O	4-19
OPEN	4-24
P	4-18
PASCAL	4-22
PR#	4-25
PRINT	4-25
R	4-17
RLEN	4-24
S	4-36
SET	4-19
STATUS	4-38
TRACE	4-34
UC	4-21
UNLOCK	4-22
V	4-20
VTOC	4-38
WHERE	4-25
WRAP	4-23
WRITE	4-17
X	4-19

CHAPTER 5
FIXCAT — By Don Worth

The FIXCAT utility may be used to perform a number of functions for all standard DOS diskettes (13 or 16 sector):

- Check a diskette's catalog and file pointers for integrity
- Check for one file overlapping another
- Produce a detailed report on the contents of a diskette
- Detect and correct errors in the VTOC or CATALOG
- Free up DOS tracks for use with files
- Search the diskette for lost files and recover them

The general way in which FIXCAT works is to make corrections only to the VTOC and catalog. Although some errors may be detected in track/sector lists or data sectors, these areas will never be modified by FIXCAT. Further, all corrections made by FIXCAT are made to an in-memory image of the catalog track, allowing the user to abort the entire operation at any time along the way. Only after all operations and checks are complete is the user given the option to write the new CATALOG track image back to the target diskette. This makes FIXCAT relatively safe for use by beginners.

The following tutorial will help you to understand FIXCAT's operation. In this tutorial you will run FIXCAT against the practice diskette to see if there are any errors in its catalog or VTOC.

FIXCAT TUTORIAL

Boot up the Bag of Tricks diskette, select the FIXCAT utility from the main menu, and follow along with the tutorial.

The first message displayed on your screen is:

```
F I X C A T   -   FIX CATALOG UTILITY

DISPLAY ON WHAT SLOT?
 0
```

Here, FIXCAT is asking you where you want its messages to be printed. If you plan to use the Apple's screen, just accept the prompt of 0. If you want to have the output go to a printer, enter the slot number of the printer.

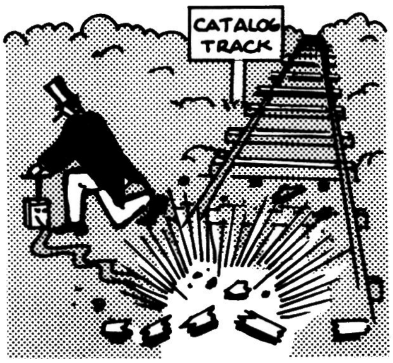
There are two things to notice about this display. One is that FIXCAT will almost always provide you with a "prompt". This means that FIXCAT is guessing what your usual response will be to each of its questions. This usual answer is printed on the input line and the cursor is left over the first letter of the prompt string. If you agree with FIXCAT, and want to use the prompt as your answer to one of its questions, just press the RETURN key. If you want to give a different answer, merely type over FIXCAT's prompt with your own. There are two reasons for having prompts. One is that this practice helps you to figure out a question by offering a typical answer. The second reason involves the AUTOMATIC TIMEOUT feature, which is discussed next.

Notice that another message appears on the current display:

```
        - TYPE END AT ANY TIME TO ABORT -
```

FIXCAT is giving you an escape hatch. Any time FIXCAT is asking you a question you may answer by typing END. FIXCAT will immediately abort everything it has done so far and return to the Bag of Tricks main menu screen.

A word of caution about the DISPLAY ON WHAT SLOT? question. If your printer interface does not echo what is being printed on the Apple screen as it prints it, you may have some difficulty in running FIXCAT with a printer. In this case you will have to look at the printer to see the text of the questions. Owners of the Apple Serial Card should set its option switches so that the 40 column with Apple video option is used (DIP switches 5 and 6 **on**). Of course, you will never have a problem if you answer "0" to this question. Do this now, if you have not already done so.



A BLOWN CATALOG TRACK MAKES A DISKETTE UNUSABLE...

The next question presented is:

```
AUTOMATIC TIMEOUT IN SECONDS?
(0 TO 255 SECONDS, 0 MEANS NO TIMEOUT)
 0
```

FIXCAT can run in two different modes. In the normal mode, each time a question is presented it will wait indefinitely for your answer. In the AUTOMATIC mode, it will wait for your answer a specified number of seconds and then, if you still haven't responded, it will assume you mean to accept its "prompt" answer. This is the other reason for having prompts on every question. Most of the time you can, with few exceptions, use all of the prompt answers, allowing FIXCAT to run by itself. For example, if you had a clobbered diskette, you could set FIXCAT up, give a 1 second timeout value for the above question, and then go do something else while FIXCAT works on your diskette. When all operations are complete, FIXCAT will wait for you to say whether you want to write the corrected CATALOG track back to your diskette. Longer timeout values may be used if you think you may want to change any answers as the questions go by, but don't dawdle! By the way, if you have set a non-zero timeout value you can stop the timer on any question by hitting a key. FIXCAT will then wait indefinitely for that (and only that) answer.

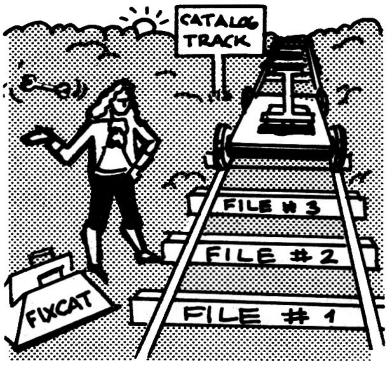
At this point, accept the AUTOMATIC TIMEOUT prompt of 0 (no timeout) by pressing RETURN. The next screen looks like this:

```
WHAT FORMAT IS YOUR DISKETTE?

13 SECTOR (DOS 3.1 OR DOS 3.2) ..OR..
16 SECTOR (DOS 3.3)

 16
```

This question helps FIXCAT to determine which RWTS to use. If you are using a 16 sector practice diskette, accept the prompt. If you are using a 13 sector diskette, type 13 and press RETURN.



...BUT FIXCAT RESTORES THE CATALOG, AUTOMATICALLY!

FIXCAT will now ask you:

```
READ EXISTING CATALOG FROM DISKETTE
OR START FROM SCRATCH? ("R" OR "S")
 R
```

It is time to insert your Bag of Tricks practice diskette (See Chapter 1) into the diskette drive. Use the drive you used to boot the Bag of Tricks diskette (you must remove the Bag of Tricks diskette, FIXCAT will not need it again unless you type END or until it completes operation).

FIXCAT is now asking you, in essence, whether you are just checking your diskette's integrity or if its CATALOG is destroyed and must be rebuilt from scratch. Ordinarily you will answer this question with "R", read the existing CATALOG. This will cause FIXCAT to read your catalog track into memory and perform all future operations on this memory image. If you respond "S", FIXCAT will start with a zeroed catalog track image. The only time you would use this latter option is when you know that the entire CATALOG track is lost (or you have already reformatted the entire track without preserving data using the INIT program).

Respond "R" (accept the prompt). FIXCAT will read the diskette's catalog track (the drive turns on) and then begin checking it for errors. Some messages will appear on the screen (probably too fast for you to see them, unless an error is found):

```
CHECKING FORMAT OF VTOC FOR VALIDITY ...

CHECKING FORMAT OF CATALOG ...
```

Then the following screen will appear:

```
DOES THIS DISKETTE CONTAIN A DOS IMAGE
ON TRACKS 0, 1, AND 2?
 Y
```

Normally the answer to this question would be yes. All standard Apple DOS diskettes contain an image of DOS on their first three tracks. If you don't plan to boot the diskette, however, you can steal 2 of these tracks back from DOS and use them for more file space (using track 0 is not reliable, since a track/sector list can not reside there). If you wanted to recover tracks 1 and 2 for your own use, you could answer N to the question (or if you had previously run FIXCAT against this diskette to do this). For more information on this procedure, see Chapter 6. In this case, our practice diskette does contain a DOS image so accept the prompt of "Y".

FIXCAT will now check each file it finds in the catalog for validity. You will see messages like these scroll by:

```
FILE: HELLO

TRACK/SECTOR LIST: T$12 S$0F
DATA: T$12 S$0E

TOTAL SECTORS ALLOCATED TO FILE: $0002
```

These are the messages for one file, HELLO (the name might be HI THERE if you have previously run through the ZAP tutorial). The location of the sector that contains the track/sector list is given as track $12, sector $0F. The contents of the track/sector list sector are then printed, forming a list of all the data sectors in the file. In this case there is only one data sector, at track $12 sector $0E. Thus, counting the track/sector list sector and the single data sector, this file has $0002 sectors. Similar messages will appear for all the other files on the practice diskette.

At this point the existing catalog and files have been checked for validity and have passed muster (if an error had occurred, an error message would have appeared). FIXCAT now asks:

```
SCAN FOR LOST OR DELETED FILES?
 Y
```

There should be no deleted files on this diskette (it was freshly INITed) and certainly no lost files. The search for lost files takes a few minutes, so we won't bother this time. Change the prompt answer to "N" and press RETURN.

FIXCAT now checks the VTOC free sector bit map against one it has been compiling while checking your files. These messages appear:

```
  72 SECTORS IN USE
 488 SECTORS FREE

NO ERRORS IN VTOC BIT MAP WERE FOUND
```

You are told how many sectors of the diskette are being used by files, the catalog, and DOS and how many are left for future use. If FIXCAT's computed map matches the one on your diskette, the NO ERRORS... message appears.

These final messages now appear:

```
        PROCESSING COMPLETED.

        NO CORRECTIONS TO CATALOG ARE REQUIRED

        PRESS RETURN TO CONTINUE
```

Since there were no errors detected during FIXCAT's operation, there is no need to write the memory image of the catalog track back to your diskette. If an error had occured, FIXCAT would have asked you if you wanted to rewrite the catalog track. Remove the practice diskette, replace it with the Bag of Tricks diskette, and press RETURN.

This concludes the FIXCAT introductory tutorial. Additional FIXCAT tutorials, in which DOS image tracks are reclaimed and a blown catalog track is reconstructed, are presented in Chapter 6.

FIXCAT MESSAGES

This section will explain the messages which can be displayed by FIXCAT along with a brief explanation of each. The messages will be covered here in the order in which they might appear in FIXCAT.

DISPLAY ON WHAT SLOT?

If you wish to have FIXCAT display its messages only on the Apple screen, enter 0 (the default). If you want them to be printed on a printer, give the slot number of your printer here.

AUTOMATIC TIMEOUT IN SECONDS?

Enter 0 (the default) if you want FIXCAT to always wait for your response to each question it asks. If you specify a number of seconds, on the other hand, each time FIXCAT prompts you for a response, it will wait that number of seconds before taking the default as your answer.

WHAT FORMAT IS YOUR DISKETTE?

If your diskette is 16 sector format, type 16 (the default). If it is an older 13 sector diskette, type 13.

READ EXISTING CATALOG FROM DISKETTE
OR START FROM SCRATCH? ("R" OR "S")

If your catalog is at all intact, specify R (the default) to have FIXCAT use it as a basis for its operations. Only if there is not a single valid sector left in the catalog should you specify "S". In this case, FIXCAT will start with a zeroed out catalog track and will build it up from scratch.

CATALOG TRACK MAY REQUIRE RE-INIT

While attempting to read your catalog track into its memory buffers, FIXCAT encountered an I/O error. If the error is merely a read error, it will be taken care of when the corrected track is rewritten to the disk. If the error is due to damaged sector formatting, then an I/O error will occur when the final write operations are attempted. In this case, it will be necessary for you to use the INIT utility first, to correct the sector formatting, before attempting to run FIXCAT again.

CHECKING FORMAT OF VTOC FOR VALIDITY ...

FIXCAT is validity checking the VTOC in memory. This could be a copy of the VTOC it read from your diskette or it could be a zeroed sector if you answered the above question "S". The following messages may appear if errors are detected.

LINK TO CATALOG BAD

The track/sector pointer to the first catalog sector from the VTOC (at offset +1, +2) is invalid. You will be asked if you want it fixed. Reply Y or N.

VERSION NUMBER BAD

The DOS version number at +3 in the VTOC is invalid for this type of diskette. If you want FIXCAT to fix it, reply Y.

BAD VOLUME NUMBER

The volume number stored at +6 in the VTOC does not match that of the diskette. Reply Y to have FIXCAT correct it.

TSL ENTRIES/TSL BAD

The number of data sectors which can be described by a Track/Sector List (TSL) is incorrect. This constant is at +$27 in the VTOC. Type Y to have it fixed.

LAST TRACK ALLOCATED BAD

The last track allocated value is not a valid track number. Type Y to have FIXCAT correct it ($11 will be used).

ALLOCATION DIRECTION BAD

The VTOC indicator of the direction of allocation is not a +1 or a -1. Reply Y to have a +1 stored there.

TRACKS PER DISK BAD

The tracks per disk value is not 35. Type Y to have it changed.

SECTORS PER TRACK

The sectors per track value is not 16 (or 13 as the case may be). Type Y to fix it.

SECTOR SIZE BAD

The sector size should be 256. It is not. Reply Y to set it to 256.

CHECKING FORMAT OF CATALOG ...

FIXCAT is validity checking all of the catalog sectors for formatting errors. The following messages may occur if an error is detected.

CATALOG LINK BAD

A track/sector pair at +1,+2 in a catalog sector does not point to the next catalog sector. The track and sector number of the offending catalog sector are printed with this message. Type Y to fix the pointer.

BAD TSL POINTER FOR FILE

A track/sector list pointer in a file descriptor entry contains an invalid track or sector number. The name of the file is displayed and you will be asked if the entry should be deleted. If you reply Y, you may still recover the file (if it really exists) using the SCAN FOR LOST FILES option of FIXCAT.

DOES THIS DISKETTE CONTAIN A DOS IMAGE
ON TRACKS 0, 1, AND 2?

If this is a standard, bootable diskette, reply Y (default). If you want to recover tracks 1 and 2 for your own use and never boot this diskette again, reply N. Also reply N if you have previously recovered these tracks using FIXCAT.

FILE: filename

FIXCAT is checking this file, found in the catalog, for validity and is using it to build its sector allocation map. The following messages may appear.

TRACK/SECTOR LIST: T$nn S$nn

FIXCAT is about to read a track sector list sector for the file. The location of the list on the diskette is given.

UNABLE TO READ FILE. DELETE IT?

FIXCAT was unable to read the first track/sector list sector because of an I/O error. If you reply Y, the file will be deleted from the in-memory image of the catalog. You may be able to recover the file later using the SEARCH FOR LOST FILES option (assuming that the TSL wasn't really where the catalog entry said it was).

DATA: T$nn S$nn T$nn S$nn ...

This is a list of the data sectors contained by this file, as described by this TSL sector.

UNABLE TO COMPLETE PROCESSING FOR FILE

FIXCAT was unable to read a subsequent track/sector list because of an I/O error. No further checking for this file will be done. This could mean that the TSL link pointer in the current TSL is bad or that an I/O error has crept into the second TSL. FIXCAT will mark the remaining data sectors of the file free for use by other files, so you may not want to rewrite the catalog track.

BAD TRACK/SECTOR POINTER

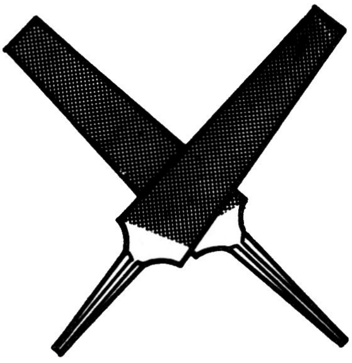
One of the track/sector pairs, describing a data sector, in the track/sector list is not valid. It is the last one printed on the DATA: message. You may have to use ZAP to correct it, assuming you can figure out what it is supposed to be.

TOTAL SECTORS ALLOCATED TO FILE: $nnnn

Given here is the count of the total number of sectors belonging to the file, including TSL sectors.

WARNING: ONE OR MORE SECTORS IN THIS
FILE OVERLAP A PREVIOUSLY PROCESSED
FILE. COPY THIS FILE TO ANOTHER DISK,
DELETE IT, AND RERUN FIXCAT.

While marking the sectors of this file "in use" in the VTOC freespace map, FIXCAT noticed that a previously processed file, or the DOS image or catalog track itself, had allocated these sectors for its use. It is not clear who got there first or how much of the file may be damaged. You should halt execution of FIXCAT, copy this file to another diskette and delete it from this one. This will free its sectors (and probably some of the other file's as well). To correct this, rerun FIXCAT against this original disk. Repeat this process each time you see the above message. You will then have to use ZAP or some other program to examine the affected files carefully for loss of data.


OVERLAPPING FILES

EXISTING SECTOR COUNT WRONG. FIX IT?

FIXCAT's count of allocated sectors for the file does not match that of the catalog entry. Type Y to change the catalog entry.

SCAN FOR LOST OR DELETED FILES?

If you do not think you have any missing files, reply N (the default). If you wish to have FIXCAT search the entire diskette for "unattached" track/sector list sectors, type Y. One word of warning, however. FIXCAT may find some old files you don't want back, such as previously deleted ones. These old files may be somewhat "moth eaten" in that some of their sectors may have been appropriated for use by other files and are overwritten. While the scan is in progress, FIXCAT will display the first data sector of each file it finds to ask you if you really want it recovered. The following messages will occur during a scan.

SCANNING DISK FOR LOST FILES

FIXCAT is reading every sector on the diskette from track 1 on for what might be a track/sector list. This takes time so be patient.

CATALOG IS FULL. SCAN HALTED

A lost file was found but there is no more room in the catalog to add an entry. The scan is prematurely stopped.

```
FILE LOCATED.  FIRST DATA SECTOR:
xxxxxxxxxxxxxxxxxxxxxxxx ............
xxxxxxxxxxxxxxxxxxxxxxxx ............
xxxxxxxxxxxxxxxxxxxxxxxx ............
etc.
```

A track/sector list has been located which has no corresponding entry in the catalog. Using the list, the first data sector has been read and some of it is displayed on the screen in hexadecimal and character. Use this display to try to identify the file so you can give it a recognizable name.

UNABLE TO DUMP FIRST SECTOR

FIXCAT got an I/O error attempting to read the first data sector of the file, so no hex/character dump could be done.

RECOVER THIS FILE?

If you think that this is a file you want, reply Y (the default). If the file looks like an old deleted file you no longer wanted, reply N. If you reply Y, the following messages may appear.

PLEASE GIVE A NAME TO THE FILE.

Using the hex/character dump, try to make up a suitable name for the recovered file. If you can't think of a name, FIXCAT will prompt you with a unique one.

WHAT TYPE OF FILE IS IT? (T,I,A,B,R,S)

If you were able to identify the file, you probably know its file type as well. If not, FIXCAT will prompt you with a pretty good guess. You'll find that FIXCAT will be correct in its guess about 95% of the time. If the wrong file type is given and you later detect this (by trying to load an Integer basic file into Applesoft, for example) you can delete the file and rerun FIXCAT, supplying a different type this time.

FILE: filename
etc.

At this point FIXCAT will process the file, exactly as it does files which were already in the catalog. See the descriptions of these messages earlier for more information about them.

nnnnn SECTORS IN USE

FIXCAT is analyzing its completed VTOC freespace map. There are nnnnn sectors allocated to files, the catalog, or the DOS boot image.

nnnnn SECTORS FREE

FIXCAT has determined that there are nnnnn sectors left for future allocation by files.

NO ERRORS IN VTOC BIT MAP WERE FOUND

The computed VTOC freespace map image in FIXCAT's buffer matches the one on your diskette.

ONE OR MORE ERRORS WERE FOUND IN THE
VTOC BIT MAP. CORRECT THEM?

FIXCAT's computation of the freespace map does not match the one on your diskette (this could be because you recovered a file or it could be your diskette has some "lost" sectors). If you want to update the VTOC freespace map with FIXCAT's corrected version, reply Y. Note that answering the DOS ON TRACKS 0, 1, AND 2 message incorrectly for your diskette will produce this message.

PROCESSING COMPLETED.

All validity checks have been completed against your diskette. FIXCAT is ready to write the corrected catalog track image back to the diskette. Up to this point it has not written to your diskette in any way.

APPLY ACCUMULATED CORRECTIONS TO THE
VTOC/CATALOG TRACK?

Reply Y if you want the corrections you have authorized up to this point to be applied to the catalog track on your diskette. Reply N if you want to forget the whole thing and not change your diskette after all.

NO CORRECTIONS TO THE CATALOG ARE REQUIRED

No changes were ever made to the memory image of the catalog track so there is no point in writing it back to the diskette.

CATALOG TRACK DOES REQUIRE RE-INIT

An I/O error was encountered while trying to write the catalog track image back to the diskette. The most likely cause is that the formatting for one or more sectors on the catalog track is damaged. You will have to run the INIT utility to correct this and then rerun FIXCAT.

ERROR (T$nn S$nn): BAD TRACK/SECTOR POINTER

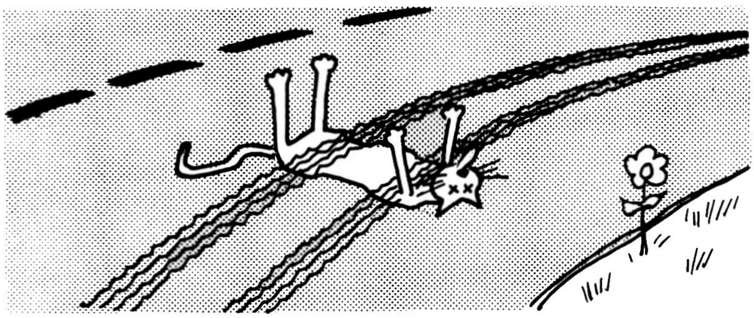
FIXCAT has attempted to read a sector whose track or sector value is out of range. The offending values are printed in hex. This message is usually associated with another FIXCAT error message, which will immediately follow it.

ERROR (T$nn S$nn): I/O ERROR

FIXCAT got an input/output error trying to read or write the track/sector indicated.

ERROR (T$nn S$nn): WRITE PROTECTED

FIXCAT was trying to write back the corrected catalog track when it determined that the write-protect notch of the diskette is covered. No write could be performed.



THIS CAT FIXCAT CAN'T FIX . . .

CHAPTER 6
ADVANCED TUTORIALS

IDENTIFYING AND CORRECTING FORMATTING ERRORS

Errors in diskette formatting are fatal. If any portion of either an address field or a data field is damaged, even a single bit out of place, DOS will not be able to read the sector in question. If the error occurs in the data field portion of the sector, the data will be lost but the sector may be reused. DOS will be able to write to that sector because it need only find the address field to perform a write operation. An error in the address field of a sector prevents DOS from writing to it, and the sector must be reformatted.

Using TRAX's verify command (V) you can quickly locate I/O Errors. Simply boot the Bag of Tricks diskette and select TRAX from the menu. After inserting the suspect diskette in the drive, press the R key to read and analyze the first track (track 0). Noting any errors, press the V key to verify the diskette. TRAX will beep and display its analysis each time an error occurs on a particular track. Make a note of the location and nature of each error and continue the verification process by pressing the V key.

If TRAX locates an error, it will be one of three types.

1. You may see the word DAMAGED in one or more locations on the screen. The line that the message is on corresponds to the sector that is damaged and you may take appropriate action. Unfortunately, while you probably can reformat the sector, the data is most likely not recoverable.
2. Another possibility is that the normal TRAX display will be generated but some portion of the display will be in inverse. The location of the information that is in inverse will tell you what part of which sector is damaged. In this case, however, the data may be recoverable and you should refer to the LOCATING AND FIXING AN I/O ERROR tutorial.
3. The third possibility is that the entire track has been damaged, in which case the message UNABLE TO INTERPRET DATA will be displayed.

To recap, if an error occurs in the data field portion of a sector, you should follow the procedure outlined in the LOCATING AND FIXING AN I/O ERROR tutorial. An error in the address field portion of a sector or a completely destroyed track require reformatting. After attempting to recover the data (see the above mentioned tutorial and the Appendix) you can use INIT to reformat the track in question. Any valid data on the track can be preserved, allowing you to, in effect, reformat the damaged sectors.

IMPROVING ACCESS TIMES FOR DOS DISKETTES

Even though much of your work on the Apple Computer is not time critical, it is nevertheless desirable to speed up execution time. Disk access time is the major problem in a great many tasks. INIT provides you with a simple way to decrease the time required to read programs from diskette. The method used is to reorder the sectors on the diskette. This procedure is referred to as **reskewing.**

By rewriting your diskette using an optimal skewing pattern you can expect between 25 and 45 percent improvement when loading Binary, Integer or Applesoft files. While reskewing a diskette doesn't speed up all operations (accessing text files is only marginally affected by skewing patterns), reskewing is simple to do and causes no compatability problems that often accompany both hardware and software products that increase the Apple's capabilities. The only difficulty with reskewing is that the copy program on the DOS 3.3 Master assumes a conventional skewing and will not perform as fast on a diskette that has been reskewed.

Boot the Bag of Tricks diskette and select the INIT program. The INIT screen display will appear (it will look like Figure 3.1 of Chapter 3). Remove the Bag of Tricks diskette and insert the diskette you wish to reskew. If you have two disk drives you may use either drive. You are going to reskew the data portion (tracks 03-34) of a diskette. You should see the cursor (>) on the top line of the data entry portion of the screen. The line will normally look like this:

```
DISK SECTORING  >16
```

Using the arrow keys, select the number of sectors that matches the diskette you have placed in the disk drive (13 for DOS 3.1, 3.2, 3.2.1 and 16 for DOS 3.3). This is important, although the program can determine if you have made an error. When you have made your selection press the space bar to move to the next line. You should see this:

```
DISK FORMAT   > DOS
```

This should be set to DOS. If it isn't, do so using the arrow keys. Then proceed to the next line using the space bar. You should see this:

```
PRESERVE DATA   > YES
```

Make sure you answer YES, because otherwise you could lose the data on your diskette. Proceed to the next line where you should see this:

```
SKEW DIRECTION   >DESCENDING
```

The last section of Chapter 3 explains why a DESCENDING skew is optimal for a DOS diskette. Move to the next line, where you should see this:

```
SKEW FACTOR   >02
```

The optimal skew is 09 and you should now make that selection using either arrow key. Move to the next line when done, where you should see this:

```
SLOT   >06
```

Select the slot number appropriate for your system (i.e., the slot your disk controller card is plugged into). Continue on and select the appropriate drive number on the next line.

```
DRIVE   >01
```

Make sure you are selecting the correct drive, then move to the next line.

```
VOLUME NUMBER   >DEFAULT
```

It would be best to use DEFAULT, which will use the volume number already on your diskette. Under some circumstances different volume numbers on the same diskette could cause a problem, so you should also reskew the first three tracks (0-2) if you wish to change the volume number from its current value. See the tutorial, CHANGING THE VOLUME NUMBER OF A DISKETTE. When ready, proceed to the next line.

```
        STARTING TRACK   >00-DEC  00-HEX
```

The correct value is 03, because the skewing you have selected does not work efficiently during the boot process. When you have changed the starting track number to three, move on to the last line.

```
         ENDING TRACK   >34-DEC  22-HEX
```

The correct value here is 34 (22 Hex), which is the last track on your diskette. Please check to see that you have made the correct entries. Your screen should match the one below. If you see any mistakes, simply use the space bar to move the cursor to the appropriate line and change the value (using the arrow keys) to its correct value.

```
              * *  I N I T  * *

          DISK SECTORING   [same as before]
             DISK FORMAT   DOS
           PRESERVE DATA   YES
          SKEW DIRECTION   DESCENDING
             SKEW FACTOR   09
                    SLOT   [your choice]
                   DRIVE   [your choice]
           VOLUME NUMBER   DEFAULT
          STARTING TRACK   03-DEC  03-HEX
            ENDING TRACK   34-DEC  22-HEX
```

When you are ready to proceed, press the RETURN key. You will then see the following prompt near the bottom of the screen.

```
      IS THE ABOVE OK ?  >YES
```

Assuming you have already carefully checked the data you entered, simply accept the YES prompt by pressing RETURN. You should now see these four lines at the bottom of the screen.

```
        DATA WILL BE PRESERVED
  INSERT DISKETTE IN SLOT 0X DRIVE 0X
      PRESS RETURN TO CONTINUE
        PRESS ESCAPE TO ABORT
```

Double check that your diskette is in the slot and drive indicated and that the door is closed. Then press the RETURN key to begin the reskewing process. You will hear the disk arm recalibrate as would proceed a normal DOS "INIT" command. Shortly thereafter you will see a prompt line indicating which track is being processed. The prompt will change to indicate which function is being performed. The whole procedure should take less than 75 seconds, at which time you will have an optimally skewed diskette. The prompt line EXIT PROGRAM ? will appear, indicating the completion of a successful initialization.

CHANGING VOLUME NUMBERS

Changing the volume number on a diskette normally requires initializing a new diskette and transferring all files from the old diskette onto the new one. Using the INIT program, changing the volume number can be accomplished much more quickly and easily.

Refer to Chapter 3 or to the preceding tutorial if you are not familiar with the INIT program. Set up the inputs to INIT as follows (this tutorial assumes a 16 sector diskette):

```
                * * I N I T * *

  DISK SECTORING   16
     DISK FORMAT   DOS
   PRESERVE DATA   YES
  SKEW DIRECTION   DESCENDING
     SKEW FACTOR   02
            SLOT   [your choice]
           DRIVE   [your choice]
   VOLUME NUMBER   [your choice]
  STARTING TRACK   00-DEC  00-HEX
    ENDING TRACK   34-DEC  22-HEX
```

If you wish to reskew your diskette in addition to changing the volume number, please see the preceding tutorial on improving access time. The normal skew is 02, but you may wish to change this value to 09 to obtain the optimal skew. Note that when you change the volume number you must change all tracks on the diskette (0 to 34), because different volume numbers on the same diskette could, under some circumstances, cause a problem. If you are also reskewing, you may wish to make two passes over the diskette, one for tracks 0-2 (skewed at 2 DESCENDING for DOS 3.3) and another for tracks 3-34 (skewed at 9 DESCENDING for DOS 3.3).

Check your input data carefully and make changes if necessary. Make sure that your diskette is in the slot and drive indicated and that the drive door is closed. Then press the RETURN key to begin the initialization process. When INIT completes, run FIXCAT, as described in Chapter 5, to fix the volume number in the VTOC.

FINDING THE A AND L VALUES OF A BINARY FILE

Sometimes it is useful to know what address (A) and length (L) values were used to BSAVE a B type DOS file. To do this you can use ZAP to examine the file.

You may even wish to modify the A value of the file. If the binary data was stored from an address different from where it is intended to be loaded, you must specify the A operand when issuing the BLOAD or BRUN command. Using ZAP, however, you can change the address in the binary file, eliminating the need to specify the A operand when loading.

Boot up the Bag of Tricks diskette, select ZAP, and then insert the practice diskette once ZAP is "up".

You can now use the OPEN command to find the binary file.

```
        OPEN'BINARY FILE'
```

In the case of the practice diskette, the first four bytes of hexadecimal displayed are:

```
        D0033000
```

This corresponds to a BSAVE command of:

```
BSAVE BINARY FILE,A$03D0,L$0030
```

Notice that the address (A) is stored in the first two bytes of the file (in reverse byte order) and the length (L) follows it, also reversed. To change the address, use this command:

```
0:D002
```

Here the address has been changed to $02D0. To finalize the change you must rewrite the sector to disk:

```
UNLOCK WRITE LOCK
```

Now, whenever BINARY FILE is BLOADed it will, by default, be loaded at $02D0.

PATCHING DOS USING ZAP

ZAP comes in quite handy whenever you want to patch DOS directly on the diskette. The DOS image which is loaded into your machine when you boot a standard DOS diskette resides on the first three tracks (0, 1, and 2). By ZAPping this image, you are changing the DOS that will be loaded when you boot the diskette. Below are several examples of how to patch DOS to do interesting things.

Each time you have identified a patch you want to make to DOS, you must first locate the patch in the memory copy of DOS and test it there. For example, suppose, after reading Beneath Apple DOS's Chapter 8 (DOS Program Logic), you have decided it would be nice to be able to remove the limitation on the L operand of the BSAVE command where only up to 32K chunks of memory can be saved (you have in mind saving 48K, for example). First you found the table of valid keyword ranges at the top of page 8-20 in Beneath Apple DOS. Then, by comparing this to the DOS in your 48K Apple, you determined that the 32767 high limit for the L keyword in this table is at $A963.

If you now look up $A900 in the DOS LOCATION TABLE on the Beneath Apple DOS reference card (under RELOCATED ADDRESS) you will see that the disk sector containing this page is on track 01, sector 08. Therefore, in order to change the 32767 value to 65535 on the disk, you must read track 01, sector 08, position to +$63 ($A**963**, remember?) and store $FFFF over the $FF7F. Then WRITE the updated sector and the patch has been applied.

In each case below, the patch addresses are given as well as the ZAP commands necessary to apply them. A process similar to the one just described can be used to make each patch. In each case the write flag is left in the UNLOCKed state, and you may wish to issue the LOCK command after all the patches you want to make are completed.

The recommended commands below include the LOG command. Whenever you patch DOS, it is recommended that you print out an audit trail of your changes by using the LOG command (if you have a printer) and keep it on file. If, at some later date, a new version of DOS comes out and you want to "migrate" the patches over to the new version, you will have a hardcopy list of what you have done to DOS.

AVOIDING RELOAD OF LANGUAGE CARD

See Beneath Apple DOS, page 7-2. $BFD3 must be changed from 8D00E0 to EAEAEA. $BFD3 is in track $00, sector $09 at +$D3.

```
LOG NOTE PATCH TO AVOID RELOAD OF LANGUAGE
  CARD
R0,9 D3 V8D00E0 D3:EAEAEA UNLOCK WRITE NOLOG
```

BRUN OR EXEC THE HELLO FILE

See Beneath Apple DOS, page 7-3. $9E42 must be changed from a $06 to either a $34 (for BRUN) or $14 (for EXEC). $9E42 is in track $00, sector $0D at +$42.

```
LOG NOTE PATCH TO BRUN THE HELLO FILE
R0,D 42 V06 42:34 UNLOCK WRITE NOLOG

LOG NOTE PATCH TO EXEC THE HELLO FILE
R0,D 42 V06 42:14 UNLOCK WRITE NOLOG
```

REMOVING THE PAUSE DURING A LONG CATALOG

See Beneath Apple DOS, page 7-3. $AE34 must be changed from a $CE to a $60. $AE34 is in track $01, sector $0D at +$34.

```
LOG NOTE PATCH TO REMOVE PAUSE DURING LONG
  CATALOG
R1,D 34 VCE 34:60 UNLOCK WRITE NOLOG
```

CHANGING THE HELLO FILE NAME

You can change the name of the HELLO file by ZAPping DOS's primary file buffer, which contains it, directly. The primary file name buffer is at $AA75. This is in track $01, sector $09 at +$75. The following commands change the name 'HELLO' to 'HI THERE'.

```
LOG NOTE CHANGE THE HELLO FILE NAME
R1,9 75 V'HELLO' 75:'HI THERE' UNLOCK WRITE NOLOG
```

PUT CURSOR ON COMMAND WHICH CAUSED A DOS ERROR

When you get a DOS error message such as "FILE NOT FOUND" or "FILE TYPE MISMATCH" because you typed the wrong file name or misspelled it slightly, it would be nice if DOS would return the cursor to the line with your faulty command so you could more easily retype it. To make DOS do this from now on, apply the following patch.

The patch must be made in two parts, one small patch in the middle of DOS and a larger one in an unused area of RWTS. The memory patches are:

```
A6FF:4C DF BC   (was 6C 5E 9D)
   and
BCDF:C6 25 C6 25 C6 25 C6 25 20 22 FC 6C 5E 9D
```

To apply this to a diskette, $A6FF is found to be in track $01, sector $05 at +$FF (the patch actually spans into sector $06 as well) and $BCDF is in track $00, sector $06 at +$DF.

```
LOG NOTE PATCH TO PUT CURSOR ON COMMAND .ND
  GIVING DOS ERROR
UNLOCK
R1,5 FF V6C FF:4C WRITE
R1,6 00 V5E9D 00:DFBC WRITE
R0,6 DF:C625 : : : :2022FC6C5E9D WRITE NOLOG
```

ALLOW THE VALUE OF THE L KEYWORD OF A BSAVE TO EXCEED 32K

This is the patch described in the introduction to this section. It allows you to save more than 32K with the BSAVE command. $A963 must be changed from $FF7F to $FFFF. $A963 is in track $01, sector $08 at +$63.

```
LOG NOTE PATCH TO ALLOW BSAVE OF MORE THAN 32K
R1,8 63 VFF7F 63:FFFF UNLOCK WRITE NOLOG
```

EXAMINING/CHANGING BASIC'S RANDOM TEXT FILES

If you use DOS from BASIC extensively, you will find ZAP a great help while developing and maintaining your text files. In this tutorial we will examine a typical random-type text file, created by a BASIC program.

To follow along with the tutorial, enter the Applesoft BASIC program shown in Figure 6.1, save it on the practice diskette, and run the program to create the file, TELEPHONE DIRECTORY. Notice that the program creates a text file of company names with their area code and phone number. The data is written in 65 byte records and is scattered around the file, leaving empty records here and there. When you type in the program, please be sure to type a CTRL D between the quote marks in line 10.

```
10 D$ = ""
20  DIM N$(20),N(20)
25  DATA  0,1,2,5,6,8,9,15,16,17,19,21,22,28,29,31,33
26  FOR I = 0 TO 16: READ N(I): NEXT I
30 N$(0) = "QUALITY SOFTWARE,213,3446599"
40 N$(1) = "ON-LINE SYSTEMS,209,6836858"
50 N$(2) = "CAVALIER COMPUTER,714,7558143"
60 N$(3) = "BUDGECO,415,6588141"
70 N$(4) = "SOUTHWESTERN DATA SYSTEMS,714,5623670"
80 N$(5) = "MICROSOFT,206,4541315"
90 N$(6) = "DATAMOST,213,7091202"
100 N$(7) = "ADVENTURE INTERNATIONAL,305,8626917"
110 N$(8) = "STONEWARE,415,4546500"
120 N$(9) = "SIR-TECH,315,3936633"
130 N$(10) = "MUSE,301,6597212"
140 N$(11) = "BRODERBUND,415,4566424"
150 N$(12) = "SUBLOGIC,217,3598482"
160 N$(13) = "RAINBOW COMPUTING,213,3490300"
170 N$(14) = "BEAGLE BROS.,714,2966400"
180 N$(15) = "THE BOOK CO.,213,3714012"
190 N$(16) = "SYNERGISTIC SOFTWARE,206,2263216"
200  PRINT D$;"OPEN TELEPHONE DIRECTORY,L65"
210  FOR I = 0 TO 16
220  PRINT D$;"WRITE TELEPHONE DIRECTORY,R";N(I)
230  PRINT N$(I)
240  NEXT I
250  PRINT D$;"CLOSE TELEPHONE DIRECTORY"
260  END
```

FIGURE 6.1 — APPLESOFT PROGRAM LISTING

Boot up the Bag of Tricks diskette, select ZAP, and, once it is up, insert your practice diskette. Now type:

```
OPEN'TELEPHONE DIRECTORY'
```

You will see a screen that looks like Figure 6.2. Notice that the first company name, QUALITY SOFTWARE (deservedly first!), is in record 0. Following it are two more data records and an empty record. You should now tell ZAP something about your file's organization. Give the record length with the following command:

```
RLEN65.
```

Now we can make references to records by their record number. First, imagine you wanted to find the record with DATAMOST's phone number. You could use the ZAP Look command like this:

```
L'DATAMOST'
```

```
Z A P - S6,1 V$01 RSA=$0000 +$00 #0 *DU1
:OPEN'TELEPHONE DIRECTORY'
00 D1D5C1CCC9D4D9A0D3CFC6D4 QUALITY SOFT
0C D7C1D2C5ACB2B1B3ACB3B4B4 WARE,213,344
18 B6B5B9B98D00000000000000 6599........
24 000000000000000000000000 ............
30 000000000000000000000000 ............
3C 0000000000CFCEADCCC9CEC5 .....ON-LINE
48 A0D3D9D3D4C5CDD3ACB2B0B9  SYSTEMS,209
54 ACB6B8B3B6B8B5B88D000000 ,6836858....
60 000000000000000000000000 ............
6C 000000000000000000000000 ............
78 00000000000000000000C3C1 ..........CA
84 D6C1CCC9C5D2A0C3CFCDD0D5 VALIER COMPU
90 D4C5D2ACB7B1B4ACB7B5B5B8 TER,714,7558
9C B1B4B38D0000000000000000 143.........
A8 000000000000000000000000 ............
B4 000000000000000000000000 ............
C0 000000000000000000000000 ............
CC 000000000000000000000000 ............
D8 000000000000000000000000 ............
E4 000000000000000000000000 ............
F0 000000000000000000000000 ............
FC 00000000                 ....
```

FIGURE 6.2 — ZAP SCREEN AFTER 'OPEN' COMMAND

```
Z A P - S6,1 V$01 RSA=$0002 +$49 #0 *DU1
:L'DATAMOST'               *** MATCH ***
00 0000000000000000CDC9C3D2 ........MICR
0C CFD3CFC6D4ACB2B0B6ACB4B5 OSOFT,206,45
18 B4B1B3B1B58D000000000000 41315.......
24 000000000000000000000000 ............
30 000000000000000000000000 ............
3C 000000000000000000000000 ............
48 00C4C1D4C1CDCFD3D4ACB2B1 .DATAMOST,21
54 B3ACB7B0B9B1B2B0B28D0000 3,7091202...
60 000000000000000000000000 ............
6C 000000000000000000000000 ............
78 000000000000000000000000 ............
84 000000000000000000000000 ............
90 000000000000000000000000 ............
9C 000000000000000000000000 ............
A8 000000000000000000000000 ............
B4 000000000000000000000000 ............
C0 000000000000000000000000 ............
CC 000000000000000000000000 ............
D8 000000000000000000000000 ............
E4 000000000000000000000000 ............
F0 000000000000000000000000 ............
FC 00000000                 ....
```

FIGURE 6.3 — THE 'LOOK FOR STRING' COMMAND

Your screen will now look like that in Figure 6.3. Notice that the buffer cursor is over the first hex byte of the name "DATAMOST". What record number is this? To find out, merely enter the STATUS command:

STATUS

The status information is now shown on the screen, and should appear like Figure 6.4. This screen indicates that you are currently positioned to record 9, on its zeroeth byte (R$000009,B$000000). The record length is set to $41 (65 in decimal), the file contains space for $23 records (or a total of $900 bytes) and the current sector is located at track $1B, sector $0C on the disk (note that your version of TELEPHONE DIRECTORY may be in a different place on the diskette but this is nothing to worry about).

```
Z A P - S6,1 V$01 RSA=$0002 +$49 #0 *DU1
:STATUS
           --- ZAP STATUS ---

FILE OPEN: TELEPHONE DIRECTORY
T$1B,S$0C R$000009,B$000000 RLEN$000041
FILE SIZE: $000023 RECS ($000900 BYTES)

BUFFER CHANGED       UPPER CASE DISPLAY
ASCII DISPLAY MODE   LOCKED - NO WRITES
NOT LOGGING          WRAP ALLOWED

$10 SECTORS/TRACK    $23 TRACKS/DISK
DOS16 DISK FORMAT

LOOK: C4C1D4C1CDCFD3D4
LOOK START: T$1B S$0E +$00
```

FIGURE 6.4 — THE 'STATUS' COMMAND

Now, suppose you wanted to find record 6. You could type:

R6

ZAP responds with the display shown in Figure 6.5. Note that the buffer cursor is pointing to the record containing information for SOUTHWESTERN DATA SYSTEMS. The N and P commands may have a different meaning then you might expect also. Type the P command now:

P

```
Z A P - S6,1 V$01 RSA=$0001 +$86 #0 *DU1
:R6
00 000000000000000000000000 ............
0C 000000000000000000000000 ............
18 000000000000000000000000 ............
24 000000000000000000000000 ............
30 000000000000000000000000 ............
3C 000000000000000000C2D5C4 .........BUD
48 C7C5C3CFACB4B1B5ACB6B5B8 GECO,415,658
54 B8B1B4B18D00000000000000 8141........
60 000000000000000000000000 ............
6C 000000000000000000000000 ............
78 000000000000000000000000 ............
84 0000D3CFD5D4C8D7C5D3D4C5 ..SOUTHWESTE
90 D2CEA0C4C1D4C1A0D3D9D3D4 RN DATA SYST
9C C5CDD3ACB7B1B4ACB5B6B2B3 EMS,714,5623
A8 B6B7B08D0000000000000000 670.........
B4 000000000000000000000000 ............
C0 000000000000000000000000 ............
CC 000000000000000000000000 ............
D8 000000000000000000000000 ............
E4 000000000000000000000000 ............
F0 000000000000000000000000 ............
FC 00000000                 ....
```

FIGURE 6.5 — THE 'READ' COMMAND WITH AN OPEN FILE

The buffer cursor moves back 65 bytes to position itself over the first byte of the record containing BUDGECO's data. Now enter the command:

R24.

Record 24 does not exist. In fact, the sector which would contain it if it did exist contains no non-empty records and, hence, was never allocated at all. There is a great big hole in the file where that sector would go. ZAP tells you this with the message:

***** NOT IN FILE *****

Any time you enter a ZAP command which would attempt to read this sector, you will see this message. To move around the sector you will have to read the next valid record following the "hole". ZAP does not contain a complete file manager, as does DOS, and can not add sectors to a file, either in the middle or at the end of the file.

Obviously, it is a simple matter to make changes to a random text file such as this using ZAP. Suppose QUALITY SOFTWARE's phone number has changed to 818 3446599. The following commands will make the change:

```
        R0 L'213' :'818' UNLOCK WRITE
```

RELEASING UNUSED SPACE IN FILES

With each type of DOS file (I,A,T, and B) it is possible that the contents of the file do not require the number of sectors allocated. For example, if you initially create an A file by saving an APPLESOFT program whose length requires 20 sectors on the diskette, and later shorten the program so that it now only needs 15 sectors, the file will not be shortened and 5 sectors will be wasted. The reason for this is that DOS will write the new version of the program over the old version in the existing file and change the length value (in the first two bytes of the file) but will not release any trailing sectors to the free sector pool. Likewise, a short text file written over a longer one will cause trailing sectors to pad the file out to an excessive length. To correct this problem the DOS manual suggests that you delete an old file before creating a new one with the same name. You can use ZAP to detect the problem as follows.

For Integer and Applesoft files, OPEN the file and examine the first two bytes of data in the buffer. These bytes represent the actual length of the program stored there. Suppose the first four bytes are:

```
        3E06
```

This means that the file is $063E bytes long. Position past the length bytes to the first byte of the program image:

```
        +2
```

Now position to the last byte of the program:

```
        +$063E-1
```

You should now be in the last sector of the file. To determine if this is the case, enter the STATUS command and compare the RSA at the top of the screen with the length of the file in bytes. Suppose the RSA is $0006 and the length of the file in bytes is $000800. If you add one to the RSA and multiply it by decimal 256 (this is equivalent to tacking two zeros onto its right side, $000700 in this case), the result should be the true length of the file. In our example, however, the STATUS gives the length as $000800. Since the two values differ by 256 ($100) bytes, one extra sector is allocated but not needed. To recover the sector for use by other files, you must LOAD the BASIC program, DELETE it from the diskette, and re-SAVE it.

If you are checking a B-type file you will find the length of the actual data at +2 in the first sector (not +0, as with BASIC programs). You can follow the same procedure outlined above (except that you will enter a +2 first) to determine if there are extra sectors attached to the file. To recover them, use the tutorial entitled FINDING THE A AND L VALUES OF A BINARY FILE to determine the address and length keyword values you must use with the BSAVE command. Then BLOAD the file, DELETE it from the diskette, and BSAVE it again.

Reclaiming extra sectors in a text file (T) is a bit harder. First the file must be OPENed. If the file has a random organization then the problem does not occur, since you may end up using the space later anyway. If the file is sequential, you can search for the end-of-file mark, a binary zero:

```
L0
```

The cursor will be positioned over the last valid byte in the file. The file only requires enough sectors to store everything up to this byte. Obviously, some bytes will be wasted to round things out to an even sector boundary. Extra sectors can be checked for in a manner similar to the I and A and B files, as described above. To free extra sectors, you must either write a program to read the file into memory, delete it, and rewrite it to disk or use ZAP and FIXCAT to free the sectors directly.

If you wish to use the latter procedure, follow these steps. Using the STATUS command, write down the track and sector number of the sector containing the last valid data byte in the file. Now CLOSE the file:

```
        CLOSE
```

Position to the VTOC:

```
        VTOC
```

and scan for the file in the CATALOG:

```
        L'MY FILE'
```

ZAP will now be positioned to the file name in the CATALOG file descriptive entry. Back up to the track/sector pointer for the first track/sector list (TSL):

```
        -3
```

and ask ZAP to read this sector:

```
        %
```

You should now be looking at the TSL for the file. See if you can find a track/sector pair that matches the sector containing the last valid data byte (you wrote it down earlier). If you see it, you can use the Look command to position the buffer cursor to that spot in the buffer. Now position to the track/sector pair describing the first extra sector:

```
        +2
```

and set the remainder of the TSL to zero (releasing the unwanted sectors):

```
        SET0
```

Now write the corrected TSL back to disk:

```
        UNLOCK WRITE
```

At this point you may think that you are done — the sectors have been deleted from the file. On the contrary, the sectors have been deleted but they are not marked free. They are "lost" sectors, not belonging to a file and not known to DOS to be free either. To recover them for use by DOS you must run FIXCAT against the diskette as described in the tutorial, RECOVERING LOST SECTORS IN THE VTOC FREESPACE MAP.

Of course, you may not want to go through all of this to recover a couple of sectors, but if you have found and released a number of sectors in various files using ZAP, you can run FIXCAT once at the very end of the process to harvest them all at once. Note that this procedure could be used on I, A, and B type files as well, but the LOAD/DELETE/SAVE or BLOAD/DELETE/BSAVE procedure is much faster.

SCANNING A DISK FOR I/O ERRORS

If you have seen the DOS message, I/O ERROR, while using one of your diskettes or if you just want to play it safe, you can use ZAP to scan your diskette for any sectors which might be damaged in some way. Once you have located an Input/Output error in a sector, you can use the next tutorial to try to fix it. (It might be a good idea for you to read the Appendix, A DISCUSSION OF DISK I/O ERRORS, before you work on this tutorial.)

To scan a disk for I/O errors, first boot the Bag of Tricks diskette, select ZAP, and then, once ZAP is up, insert the diskette to be checked. Start by reading track 0, sector 0:

```
R0,0
```

Next, type the following line:

```
NOWRAP N LOOP
```

This command line will turn off the WRAP option so that the scan will stop when the last sector on the diskette has been read (track $22, sector $0F). If you didn't do this, the command line would loop forever! The N command will read the next sector, and the LOOP command will repeat the process. The entire operation will take a minute or two. If there is an I/O ERROR on your diskette, the command will be halted and an error message will be displayed. If this happens, write down the track and sector number of the error and retype the command line. If, instead, the command ends with the message 'NUMBER TOO BIG' on track $22, sector $0F, then there were no I/O errors.

This method can also be used to check for I/O errors within a file. Just OPEN the file first and then follow the above steps.

If you have detected one or more I/O errors on your diskette, use the following tutorial to try to get around it.

LOCATING AND FIXING AN I/O ERROR

Imagine that you have just seen a DOS I/O ERROR message and have run the previous tutorial to locate it. Now it is time to try to do something about it. First you will determine whether the I/O error is intermittent. Try reading the faulty sector again (you should have noted the track and sector number of the error when you scanned the diskette. If you haven't done this, run through the SCANNING A DISK FOR I/O ERRORS tutorial before working on this one). Use the ZAP Read command to try to read the sector. If you still get an I/O error, keep trying several times. You can use some of the techniques listed in the Appendix as well.

If you finally manage to read the sector, the first thing to do is save the data somewhere else. WRITE it to track $02, sector $0F. This is an unused DOS image sector. Of course if you have freed track $02 in order to reclaim it from DOS for your own use (using FIXCAT or a similar utility) you will have to save the data somewhere else. Now try to reWRITE the bad sector. Many times, the error will go away if the data is rewritten. If this too produces an I/O error, you will have to use the INIT utility to correct the formatting for this sector. See the tutorial entitled IDENTIFYING AND CORRECTING FORMATTING ERRORS for information on how this is done. If you succeed in writing the data back, try rereading it a number of times. If no more error messages occur then you have probably fixed the error. In any case, it might be a good time to copy the diskette to a new one and put the old one "out to pasture".

If you can't read the sector data then it is probably not recoverable. You should, at this point, try to determine whether the sector formatting is gone, along with the data. Try WRITEing to the sector:

```
        SET0 UNLOCK WRITE
```

If this produces an I/O error, you will have to use INIT to reformat the sector. Refer to the aforementioned tutorial, IDENTIFYING AND CORRECTING FORMATTING ERRORS. If you did manage to write the zeroed buffer, you will want to know what part of your diskette has been clobbered.

If you are on track $00, $01 or $02 you have probably stepped on the DOS boot image. This can be fixed by running the Apple utilty, MASTER CREATE. If you have stepped on track $11, the CATALOG has been damaged. You will have to run FIXCAT to recover what has been lost. Refer to the tutorial, RECONSTRUCTING A BLOWN CATALOG. If you are positioned to any other track you can use the WHERE command to find out what file has been damaged:

```
        WHERE
```

After a while, ZAP will respond with an I/O ERROR. Type STATUS to find out what file has been opened. If you were positioned in an area not contained by any file, the message "NOT FOUND" will be displayed. In this case you are safe! If a file does contain the I/O error, there is a possibility that the sector is not being used. The bad sector could be extra padding at the end of a file which has been shortened. Read through the tutorial, RELEASING UNUSED SPACE IN FILES, for more information on this. In this case, the INIT utility can be used at once to correct the sector formatting without fear of data loss.

If the bad sector is in a file, you will want to try to patch it back together, however. If the file is a Text file, look at the sectors immediately before and after the bad one (use the P and N commands). Perhaps you will be able to figure out what was in the lost sector and ZAP it back. If not, you can at least change it to something more intelligible than zeros. A good choice for a text file would be hex $8D's (carriage returns). This will have the effect of inserting 256 null length records in the file. You can then use BASIC to read and recover the data after the damaged sector by skipping these records.

If the file is a B, A or I type file you may have to kiss it goodbye. Unless you are adept at BASIC's tokenizing, you may have a tough time putting something into the sector which won't cause BASIC to choke. Of course it is always worth a try. Load the program into BASIC and see if you can piece it back together. Likewise, it is pretty hard to remember what was in a binary file, since a sector can represent over a hundred instructions.

COPYING PASCAL FILES TO DOS USING ZAP MACROS

This tutorial will demonstrate how to copy an Apple PASCAL text file to a DOS text file. This combination has been selected merely as an example. In principle, you should be able to apply this technique with very minor changes to the copying of 13 sector to 16 sector or vice versa, or any combination of CPM, PASCAL or all versions of DOS. Any file that ZAP can OPEN can be copied to any other file ZAP can OPEN.

The procedure used here involves two disk drives. It is possible to use ZAP to copy files using one drive, but the tedium involved does not lend itself well to a tutorial. If you have only one drive, read through this tutorial and make sure you understand what is involved. It should be a relatively simple matter for you to modify this procedure to use one drive (you will have to enter commands prior to each disk swap). Another approach might employ ZAP's multiple buffers to read up to 16 sectors in before switching diskettes.

In this example we will copy the PASCAL file GRAFCHARS.TEXT. This file appears on one of the Apple PASCAL system diskettes, APPLE3:. If you do not have PASCAL for your Apple, you will have to follow along with this tutorial without actually performing it or you can substitute a CPM or DOS input file and change all references to the file name and the diskette type in the ZAP commands given.

The first thing to do is to create the output file. ZAP does not contain any file management to speak of, so it can not create a file or add sectors to an existing file. You will have to pre-create the file with at least as many sectors as the input file. Don't worry if you make it too big. You can always shorten it using the procedure discussed in the tutorial, RELEASING UNUSED SPACE IN FILES. We will start by OPENing the PASCAL file to find out how large it is. Boot up the Bag of Tricks diskette, select ZAP, and, after it is up, replace the Bag of Tricks diskette with the PASCAL diskette, APPLE3:

```
        PASCAL OPEN"GRAFCHARS.TEXT" STATUS
```

After you have entered the above command to ZAP, you will see a STATUS screen with a value for the FILE SIZE in records (it is $00000C). There are 12 records in the file ($0C is hexadecimal for 12). Now replace the APPLE3: diskette with a diskette that has DOS on it, boot DOS, and insert your practice diskette in the drive. Enter the following command to force DOS to create an output file with enough sectors:

```
BSAVE GRAFCHARS,A$800,L$0D00
```

Notice that the length specified on the BSAVE command must be the number of records in the file +1 (given in hex, $0D = 13 decimal), followed by two zeros. Now that the output file is created, reinsert the Bag of Tricks diskette and rerun ZAP. The first thing to do is to change the file type of the DOS file, GRAFCHARS, from B (binary) to T (text). Enter the following commands in ZAP, after inserting the practice diskette in the drive:

```
VTOC L'GRAFCHARS' -1
```

You should be positioned one byte before the file name in the CATALOG entry describing the file, GRAFCHARS. This byte is the file type and should currently be a $04 (Binary file). Change it to a $00 (Text file):

```
:00 UNLOCK WRITE
```

Be sure to type the colon. Now OPEN the file and set it to zeros:

```
OPEN'GRAFCHARS'
UNLOCK NOWRAP SET0 WRITE N LOOP
```

When you see the message, NUMBER TOO BIG, the entire file has been set to zero. You are ready now to enter the ZAP macros you will need to copy the file. To make room for the macros you will have to delete some of the built-in macros first. Enter:

```
/MREAD /MWRITE
```

and then:

```
(OP1 #0 CLOSE S,1 PASCAL OPEN"GRAFCHARS.TEXT")
(OP2 #1 CLOSE S,2 DOS16 OPEN"GRAFCHARS")
(START OP2 =SV2 OP1)
(COPY =SV1 OP2 #0 ATSV2 WRITE ATSV2+256. =SV2 OP1
  SV1)
```

These are all the macros you will need. If you wish, you can double check to be sure you typed these macros in correctly by issuing the MACROS command. To start the copy operation, insert the input diskette (APPLE3:) in drive 1 and the output diskette (your practice diskette) in drive 2. Now type:

```
START
```

Ugh! What is that junk? PASCAL likes to store some binary information about the file in the first few sectors. You will want to skip over that so type:

```
N
```

Keep typing N until you see the first sector containing text (that will be RSA$00004). Now you want to copy that over as the first sector of the DOS output file:

```
COPY
```

The deed is done! Now type N again to view the next sector of the PASCAL file. It looks good so type COPY again. If you are getting bored, you can copy the rest of the PASCAL file by typing:

```
NOWRAP N COPY LOOP
```

The entire file will have been copied when you see the message, NUMBER TOO BIG. At this point you can view the output file by typing:

```
OP2
```

You may now want to use the methods described in RELEASING UNUSED SPACE IN FILES to truncate the file to its actual size. It is also worth mentioning that the format of a PASCAL text file is somewhat different than that of a standard DOS file. Each record is terminated by a carriage return, as with DOS, but each record is preceeded by a $10 as well. Also, the most significant bit (MSB) of each byte is turned off — a carriage return is $0D, not $8D. You may want to use ZAP's Or command to turn on the high order bits (O80 LOOP256. WRITE) and you may want to change all occurrences of $10 to $A0 (blank). This can be done with the Look command in a loop:

```
:A0 WRITE
L10 :A0 WRITE NOWRAP LOOP
```

COMPARING FILES USING ZAP MACROS

This tutorial will show you how to use ZAP to compare two DOS files which both reside on the same diskette. Of course, the macros could be modified to operate using two diskettes and/or different diskette types. To prepare for the tutorial, boot a diskette with DOS on it and insert the practice diskette into your drive. Enter the following DOS commands:

```
BSAVE FILE1,A$800,L$1000
BSAVE FILE2,A$800,L$1000
```

These commands create two identical files of 17 sectors each. Now boot the Bag of Tricks diskette and invoke ZAP. Insert your practice diskette in the drive and enter the following ZAP commands:

```
/MREAD /MWRITE
(OP1 #0 OPEN'FILE1')
(OP2 #0 OPEN'FILE2')
(MRD #+1 AT *+256. LOOP7,-18.)
(CMP OP1 AT SV1 MRD =SV1 OP2 AT SV2 MRD =SV2 #7 #+1
  COMPARE-8 LOOP8,-18.)
```

This sets up the macros needed to do the comparison. Briefly, the MRD (multiple read) macro is called as a sub-macro by the CMP macro. MRD reads the next seven sectors into the next seven ZAP buffers. The CMP macro first opens FILE1, repositions to its remembered last location in that file (SV1) and reads that sector along with the following seven into buffers 0 through 7. CMP then performs a similar operation with FILE2, reading its sectors into buffers 8 through F. A loop is then performed to compare buffer 8 (#7 #+1 = #8) to buffer 0 (COMPARE-8), 9 to 1, A to 2, etc. If any of the COMPARE operations fail, an error message will be printed and the macro will halt. The CMP macro will have to be manually invoked for each 8 sectors to be compared.

To initialize things for the operation, perform the following commands:

```
OP1 =SV1 OP2 =SV2
```

By doing this, you have initialized the label variables SV1 (save my place in file 1) and SV2 (save my place in file 2) to point to the first sector of each file. You can now compare 8 sectors by typing:

```
CMP
```

When this macro finishes, you will be looking at the eighth sector of FILE2 (RSA $000007) in buffer $0F. In other words, the first eight sectors of the files match exactly (we already know that, of course). To compare the next seven sectors (the macro overlaps and compares sector $000007 again) type:

```
        CMP
```

again. This time you are looking at RSA $00000E. Type CMP one last time. You are now looking at RSA $000004. This means that the macro has wrapped around at the end of the files and re-compared the first five sectors of each file again. Obviously the files match.

Now position to RSA $000005 of FILE1:

```
        OP1 R5
```

and change something in the sector:

```
        3E:'JUST TESTING' UNLOCK WRITE
```

Now repeat the above operation:

```
        OP1 =SV1 OP2 =SV2
        CMP
```

The first invokation of the CMP macro ends prematurely with a "DOES NOT MATCH" error message. The buffer cursor should be pointing at offset $3E. You do not see the text 'JUST TESTING' because you are looking at the sector which was read from FILE2. Type #-8 to see the corresponding sector from FILE1 again.

This set of macros has two disadvantages. One is that CMP does not stop the compare operation at the end of the file. The other problem is that once a mismatch has been found, it is difficult to start the compare operation going again at the point where it left off. The reason both of these problems exist is the fact that multiple sectors are read at a time into all 16 ZAP buffers. If you were to redesign the macros to read and compare one sector at a time (as is done in the copy macros given in the previous tutorial) you could circumvent both problems, but the operation would take longer because both files would be reopened for every sector compared. The tradeoff is improved performance versus ease of use, and it is up to you to choose a method that best suits your needs.

PATCHING A MACHINE LANGUAGE PROGRAM USING ZAP

ZAP can be very useful to the assembly language programmer who wishes to apply patches directly to his programs using ZAP. Often, it is inconvenient to reassemble a program to make a very small change. Likewise, a complex program may not lend itself to being patched in memory and reBSAVEed (especially if it runs where DOS normally is loaded). The following tutorial will demonstrate the use of ZAP in patching a machine language program.

First, boot the Bag of Tricks diskette, select ZAP, and then insert the practice diskette. If you have a printer, turn on logging as follows:

```
        LOG
```

this will provide you with a permanent record of the changes you make to the program. Now open the practice binary file:

```
        OPEN'BINARY FILE'
```

You should now be looking at the first sector of the file (the only sector, in this case). Since we know that a binary file always consists of a 2 byte address and a 2 byte length followed by the binary memory image, you must position past these to the program itself:

```
        +4
```

Assume that this is the start of a machine language program called, appropriately enough, "PROGRAM". Cause ZAP to memorize this location in your file:

```
        =PROGRAM
```

Suppose that, after consulting your assembly listing, you wanted to examine an instruction at +0009 from the entry point, PROGRAM. You would type:

```
        PROGRAM+9 I
```

Here you see a JMP instruction:

```
        500A:4C B5 B7     JMP     $B7B5
```

followed by a number of other instructions. The JMP instruction will appear in a location in memory other than $500A when the program is BLOADed, but this just happens to be its current memory location in ZAP's buffer, which starts at $5000. If you wanted to dump these instructions to your printer, you could do so as follows:

```
PROGRAM+9 IDUMP
```

Instead of just 20 instructions, the entire remaining portion of the ZAP buffer is disassembled to the printer. Now imagine that you wish to change this instruction to JMP to $77B5 instead of $B7B5. To do so you position the buffer cursor to the instruction and store the new object code there:

```
PROGRAM+9:4CB577 UNLOCK WRITE
```

Now redisplay the instruction to see the change you have made:

```
PROGRAM+9 I
```

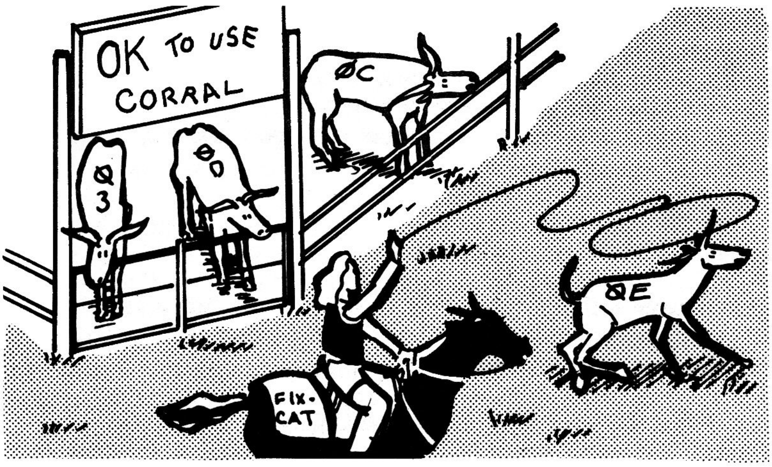
Do you see the instruction at $502C? Suppose you wanted to find it in your assembly listing. To do so you need to know its displacement (offset) in hex from the entry point. To ask ZAP to calculate this, enter:

```
2C ?*-PROGRAM
```

This gives $000028 as the displacement from the entry point to this instruction. If the instruction you wanted was in another sector of the file you could still perform this operation, but you would also have to calculate the difference in the RSA between the sector containing the instruction and the first sector of the program. This difference would form the high order byte of the offset. For example, if the instruction we just examined was actually in RSA $000A of the file then the displacement to the instruction would be $0A28.

RECOVERING LOST SECTORS IN THE VTOC FREESPACE MAP

Upon occasion it is possible that a few sectors on a diskette will become lost. These lost sectors are not marked free for use in the VTOC bit map, yet they are neither part of a file, DOS, or the CATALOG track. This can happen, for example, if you press RESET or turn the machine off while a file is being written.



RECOVERING LOST SECTORS IN THE VTOC FREESPACE MAP

When lost sectors exist, you can use FIXCAT to locate these sectors and release them for use again by updating the VTOC freespace bit map. One word of warning, however. Some media surface analysis programs will intentionally create lost sectors when it is determined that they can never be successfully formatted. If you have used a surface analysis program to prepare your diskette for use, do not use FIXCAT to recover these sectors. I/O errors will result if you do.

To check a diskette for lost sectors, boot the Bag of Tricks diskette and select FIXCAT. Insert the diskette to be checked (try the practice diskette first, although there should be no lost sectors on that one!) and follow the introductory tutorial given in Chapter 5. If there are lost sectors, FIXCAT will respond:

```
nnnnn SECTORS IN USE
mmmmm SECTORS FREE

ONE OR MORE ERRORS WERE FOUND IN THE
VTOC BIT MAP.  CORRECT THEM?
```

If you reply Y to this question, FIXCAT will correct it's in-memory image of the VTOC to mark your lost sectors free again. Now, when the message:

```
APPLY ACCUMULATED CORRECTIONS TO THE
VTOC/CATALOG TRACK?
```

appears, reply Y to have FIXCAT write its corrected VTOC back to your diskette.

RECLAIMING TRACKS 1 AND 2 FOR FILESPACE

Unless you really like the convenience of being able to boot DOS from any of your diskettes, you can increase the storage capacity of a diskette by 32 sectors (or 26 for 13 sector formats) by releasing to the free sector pool the sectors which normally hold the DOS boot image in tracks 1 and 2. Once you do this to a diskette you can not boot DOS from it but must boot another diskette (such as your system master) and then insert the modified diskette once DOS is loaded. Since files will be allocated using the newly released sectors, you should not run MASTER CREATE against this diskette as it would overwrite some of your files. Instructions are given in the next tutorial, A DOS-LESS BOOT PROGRAM, for placing a short program on track 0 which will remind you that there is no longer a DOS on the diskette should you try to boot from it.

Note that it is not a good idea to free sectors in track 0 for use with files. Although some magazine articles and utilities have advocated this, you can run into problems if DOS attempts to allocate a track sector list for a new file on track 0. In this case, the catalog entry for the file will look to DOS like an end-of-catalog marker (DOS checks the TSL track byte for zero to see if the end of the catalog has been reached), and you will never be able to access the file again. Likewise, suggestions have been made to use some of the sectors of the catalog track for file storage. The number of sectors gained in this way is usually not worth the effort (since you must still provide at least a VTOC sector and one catalog sector, leaving only 14 possible new sectors) and the practice severely limits the number of files which can be stored on the diskette. For these reasons, Bag of Tricks only supports recovery of tracks 1 and 2 as described below.

Boot the Bag of Tricks diskette, select FIXCAT, and then insert your practice diskette when requested. Accept all prompts (except, perhaps, the 13 versus 16 sector prompt) until this message appears:

```
DOES THIS DISKETTE CONTAIN A DOS IMAGE
ON TRACKS 0, 1, AND 2?
```

Since you no longer care about the DOS image on these tracks, reply N to this question. FIXCAT will still force track 0 to be allocated, since track/sector lists may not appear on track 0, but it will release all the sectors on track 1 and track 2 for DOS's subsequent use. Later, FIXCAT will display this message:

```
ONE OR MORE ERRORS WERE FOUND IN THE
VTOC BIT MAP. CORRECT THEM?
```

This results from the fact that the new freespace bit map does not match the old one (in which tracks 1 and 2 were marked allocated). Reply Y to this question to accept FIXCAT's changes to the VTOC bit map. Now, when the message:

```
APPLY ACCUMULATED CORRECTIONS TO THE
VTOC/CATALOG TRACK?
```

appears, reply Y to have FIXCAT update the diskette itself. The procedure is complete. Remember that whenever you run FIXCAT against this diskette in the future you must reply N to the question about whether DOS exists on tracks 0, 1, and 2 on this diskette. Now follow the next tutorial to place a reminder boot program on track 0 of your diskette.

A DOS-LESS BOOT PROGRAM

There have been many articles written about how diskette storage can be increased by removing DOS from the diskette and using at least two additional tracks for data storage. As pointed out in the INIT tutorial (Chapter 3), you can very easily produce a DOS-less diskette suitable for data storage. By using our INIT program you have the added benefit of a diskette optimally skewed for faster access time.

There is a drawback, however, to using diskettes solely for data. If you do try to boot the diskette, it will either spin forever or cause your disk drive to recalibrate, depending on how DOS was purged from the diskette. Below you will find a simple way to avoid that problem, a way that will work effectively on any 16-sector DOS-less diskette as long as sector 0 is unused. Unfortunately, DOS versions 3.2.1 and earlier use an encoding technique that is difficult to reproduce. For that reason this program will only work for 16 sector diskettes.

When a diskette is booted the software on the disk controller card reads track 0 sector 0 into memory and then gives control to that software, which normally carries on the boot process. We have written a short machine languge program that, rather than booting, turns off the disk drive and then prints a message on the screen informing you that DOS is not present on the diskette. The program has been written such that you may easily enter any message you wish to appear on the screen when the diskette is booted. The program itself consists of 36 bytes and may be followed by any ASCII string of 218 characters or less, terminated by a zero (00).

ZAP provides a very quick means of writing the software to track 0 sector 0. Boot the Bag of Tricks diskette and select ZAP from the menu. When ZAP has displayed its intial screen, insert your already formatted DOS-less diskette in your disk drive. Since ZAP defaults to 16 sector format you may proceed to select the appropriate slot and drive values you wish to use.

Read track 0 sector 0 into memory to make sure it is accessible. This is done with the command R0,0. In a moment the screen will display the contents of the sector. The next step is to set the entire sector to 00's, this is done with the command SET0. The sector data should rapidly change to 00's. Now you can enter the short program that follows, making sure that you type the preceding colon which tells ZAP that you are entering data at the current cursor position.

```
:01A62BBD88C02058 (return)
:FCA9258500A90885 (return)
:01201808A900F0FC (return)
:A000B100F00620F0 (return)
:FDC8D0F660       (return)
```

Your screen should now look like Figure 6.6. Please verify that you have entered the data correctly before proceeding.

The cursor is now correctly positioned to enter your ASCII string. The sample below is given to show how to enter the data. You may enter different text if you want to; just make sure it fits within the remainder of the sector. The 8D entry causes a return when the message is printed, forcing the following text to the next line.

```
:'DOS-LESS DISKETTE' (return)
:8D (return)
:'USE ANOTHER DISKETTE' (return)
```

```
Z A P - S6,1 V$FE T$00 S$00 +$25 #0 *DU1
::FDC8D0F660
00 01A62BBD88C02058FCA92585 .&+=.@ X\)%.
0C 00A9088501201808A900F0FC .)... ..).P\
18 A000B100F00620F0FDC8D0F6  .1.P. P]HPV
24 600000000000000000000000  ...........
30 000000000000000000000000 ............
3C 000000000000000000000000 ............
48 000000000000000000000000 ............
54 000000000000000000000000 ............
60 000000000000000000000000 ............
6C 000000000000000000000000 ............
78 000000000000000000000000 ............
84 000000000000000000000000 ............
90 000000000000000000000000 ............
9C 000000000000000000000000 ............
A8 000000000000000000000000 ............
B4 000000000000000000000000 ............
C0 000000000000000000000000 ............
CC 000000000000000000000000 ............
D8 000000000000000000000000 ............
E4 000000000000000000000000 ............
F0 000000000000000000000000 ............
FC 00000000                 ....
```

FIGURE 6.6. — THE DOSLESS BOOT PROGRAM

A zero must conclude the text to signify the end of the string, but since you previously set the buffer to zeroes you need not enter it.

When you are satisfied that you have entered the information correctly, you may write the buffer to your diskette. This is done by first typing UNLOCK followed by a carriage return, which enables ZAP to write. Then type the command WRITE followed by a carriage return. You have now stored the program on the diskette.

To repeat this task, you do not need to retype the information in. You can simply use ZAP to read track 0 sector 0 from an already modified diskette, insert the new diskette and write the sector onto the diskette. If you wish to change the screen message, position the cursor to the beginning of the ASCII string by typing 25 followed by a carriage return. Following the example above, type the new string into the buffer making sure to conclude it with a 00 and write it to your diskette.

UN-DELETING A FILE

There are two ways to un-DELETE a file you have accidentally DELETEd. If you have only just deleted it, and you have not created or written any other files on the diskette in the meantime, you can recover it using ZAP. The following example shows how this can be done.

Boot any diskette with DOS on it, then insert the practice diskette. DELETE the file BINARY FILE. Now boot the Bag of Tricks diskette and select ZAP. Insert the practice diskette and type:

```
        DOS16
```

or DOS13 as the case may be. This will position you to the catalog. Now type:

```
        VTOC L'BINARY FILE'
```

This will find the catalog entry for BINARY FILE. The entry is still there even though the file has been deleted. The buffer cursor will be positioned over the first byte of the file name. Now back up the cursor to the start of the catalog entry:

```
        -3
```

The cursor should be positioned over a $FF. This indicates that the file has been deleted. The $FF must be replaced with the track number of the first Track/Sector List for the file. Luckily, DOS always saves this number in the last byte of the file name before stomping on it with the $FF. To find it, type:

```
        +32.
```

Suppose that the number here is $13. So change the $FF to $13:

```
        :A0 -33. :13 UNLOCK WRITE
```

These commands tell ZAP to change the last byte of the file name back to a blank ($A0 in hex), back up to the TSL track byte, store the track number of the first TSL, and rewrite the catalog sector. Now run FIXCAT as outlined in the tutorial, RECOVERING LOST SECTORS IN THE VTOC FREESPACE MAP, to mark all of the sectors belonging to the deleted file as "in use".

If you have created new files since you DELETEd the file, there is a possibility that the above procedure will not work for you. The catalog entry which once held the name of the file you want may have been reused for another file. In this case, the Look command will not find the name in the catalog. You will have to use FIXCAT to search the diskette for the file as described below. If you do find the catalog entry for the file, there is still a chance that some of its sectors have been reused by another file. This will be indicated by the FIXCAT message:

```
WARNING: ONE OR MORE SECTORS IN THIS
FILE OVERLAP A PREVIOUSLY PROCESSED
FILE. COPY THIS FILE TO ANOTHER DISK,
DELETE IT, AND RERUN FIXCAT.
```

If this message appears, some of the sectors for your file have been used by other, newer files. You will have to copy your un-deleted file to another diskette, DELETE it from this one, and rerun FIXCAT against the original diskette to resolve the VTOC freespace conflicts. You can then see what can be salvaged by examining the copy on the other diskette using ZAP. Refer to the tutorial on LOCATING AND FIXING AN I/O ERROR for ideas on what to do with the "moth holes" in your file.

If there is no catalog entry for your file all is not lost. You can ask FIXCAT to search the diskette for the first Track/Sector List of the "lost" file (if it hasn't been overwritten) and create a new catalog entry for the file. During the process, FIXCAT will warn you if there are overlapping sectors. If so, you should follow the procedure outlined above.

To use FIXCAT to search the diskette, boot the Bag of Tricks diskette and select FIXCAT. Accept all of the prompts (as appropriate) and insert the diskette to be searched when FIXCAT asks for it. (You can try this out on the practice diskette by DELETEing BINARY FILE again.) When the message:

```
SCAN FOR LOST OR DELETED FILES?
```

appears, reply Y. FIXCAT will search the diskette for Track/Sector Lists which do not appear as part of any file described by the catalog track. If FIXCAT finds a TSL you will see something like this:

```
FILE LOCATED.  FIRST DATA SECTOR:
xxxxxxxxxxxxxxxxxxxxxxxx ...........
xxxxxxxxxxxxxxxxxxxxxxxx ...........
xxxxxxxxxxxxxxxxxxxxxxxx ...........
xxxxxxxxxxxxxxxxxxxxxxxx ...........
etc.

RECOVER THIS FILE?
```

If you think this might be your long lost file, reply Y. When the message:

```
PLEASE GIVE A NAME TO THE FILE.
```

is displayed, enter the name of your deleted file. Also respond with its type (T, I, A, or B) when prompted. FIXCAT will create a catalog entry for the file and, when you see the message:

```
APPLY ACCUMULATED CORRECTIONS TO THE
VTOC/CATALOG TRACK?
```

reply Y to finish the un-delete process. Note that you will probably see the message:

```
ONE OR MORE ERRORS WERE FOUND IN THE
VTOC BIT MAP.  CORRECT THEM?
```

This is normal. FIXCAT has reallocated all of the sectors occupied by your file and the VTOC on the diskette will not reflect this. Reply Y to this question so that the un-deleted file will not be overwritten by other files.

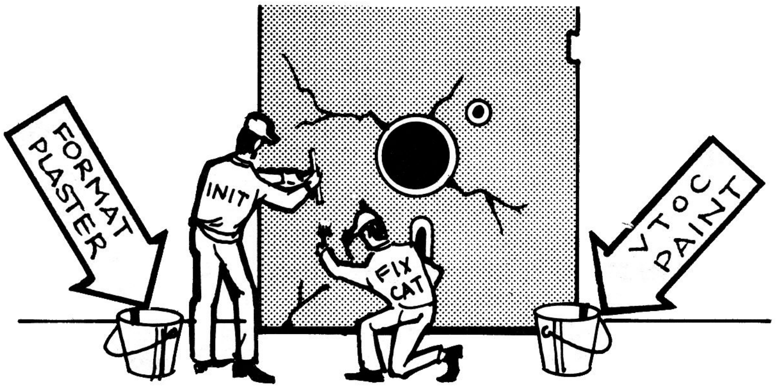
RECONSTRUCTING A BLOWN CATALOG

The surest sign of a damaged catalog track is an I/O ERROR message from DOS when you try to CATALOG the diskette. If this happens, you will be unable to do much with the diskette. The problem could be just one damaged sector, or the entire track could be gone. In either case, FIXCAT will come to your rescue and automatically recover all of your files.

This tutorial will be in the nature of a demonstration of FIXCAT's capabilities to recover lost catalogs. To perform the tutorial it will be necessary first to create a damaged catalog on the practice diskette. To do this, boot the Bag of Tricks diskette, select ZAP, and then insert the practice diskette once ZAP is up. Now type:

```
VTOC
SET0 UNLOCK WRITE N LOOP16.
```

The first line will position you to the VTOC of the practice diskette. The next commands set each sector of the catalog track, including the VTOC, to zeroes. This is an extreme case. If you were working with an actual damaged diskette you would want to save as many of the sectors as could be read. To do this you would run the INIT utility to reformat track $11 (the catalog track), preserving any data already on it. If INIT can read any of the catalog sectors it will leave them in place. Otherwise, it will write unreadable sectors back to the diskette as zeros. For more information on using INIT to reformat, read the tutorial IDENTIFYING AND CORRECTING FORMATTING ERRORS. If INIT is unable to format the diskette due to a physically damaged media, you may have to use it to copy every sector over to another diskette. (See the Appendix for patches to DOS's copy program that will allow you to do this.) This new diskette can then be operated upon as described by this tutorial.



INIT & FIXCAT WORK TOGETHER TO REPAIR DISKETTES

Now that you have a diskette with a totally destroyed catalog track, it is time to run FIXCAT. Boot the Bag of Tricks diskette and select FIXCAT. Follow along with the tutorial, using your practice diskette when FIXCAT calls for it.

FIXCAT will first ask for a slot number for output. Accept the prompt of 0 (the screen). When asked for the automatic timeout value, reply 0 (no timeout). Next reply with the number of sectors per track (16 or 13) for the target diskette. Finally, accept the prompt of R (read catalog track) to start things going. You could have said S here, and FIXCAT would not have bothered to read the catalog from the diskette (it is zero anyway), but it never hurts to read the catalog in case any of the sectors are still good. The next message you see is:

```
CHECKING FORMAT OF VTOC FOR VALIDITY...

LINK TO CATALOG BAD
FIX IT?
 Y
```

FIXCAT is now attempting to make sense out of a zeroed VTOC. By the time it has finished this phase it will have found errors in practically every field in the VTOC. The result will be a valid, albeit empty, VTOC. Reply Y (accept the prompt) on each "FIX IT?" question until you see the messages:

```
CHECKING FORMAT OF CATALOG...

T$11 S$0F CATALOG LINK BAD
FIX IT?
 Y
```

Now FIXCAT is trying to put together an empty catalog from your zeroed track. It will find that every sector requires a track/sector link to the next so reply Y to every question until you see:

```
DOES THE DISKETTE CONTAIN A DOS IMAGE
ON TRACKS 0, 1, AND 2?
 Y
```

If you have never "wiped" DOS from this diskette (as described in a previous tutorial), reply Y (accept the prompt). Otherwise, reply N. Now you will see this question:

```
SCAN FOR LOST OR DELETED FILES?
 N
```

Change the prompt to Y and press RETURN. This is the reply which causes FIXCAT to recover your catalog. FIXCAT will now spend some time searching every sector on the diskette which might contain a file's track/sector list. When it finds a TSL it will display something like this:

```
          FILE LOCATED.  FIRST DATA SECTOR:
          090007080A0080000000084A ...........J
          000000000000000000000000 ............
          000000000000000000000000 ............
          000000000000000000000000 ............
          000000000000000000000000 ............
          000000000000000000000000 ............
          000000000000000000000000 ............
          000000000000000000000000 ............
          000000000000000000000000 ............
          000000000000000000000000 ............
          000000000000000000000000 ............
          000000000000000000000000 ............
          000000000000000000000000 ............
          000000000000000000000000 ............
          000000000000000000000000 ............
          000000000000000000000000 ............
          000000000000000000000000 ............

      RECOVER THIS FILE?
       Y
```

This is a hexadecimal and character dump of part of the first sector of the file FIXCAT has located. The reason the sector is dumped is to give you a chance to try to identify the file and determine whether you want to recover it or not. It could be that the file is a previously DELETEd one which you no longer wanted. By recovering it you may create conflicts and overlapping sectors with other files. In this case we know that no files were ever deleted, so accept the prompt of Y. Now you will see:

```
      PLEASE GIVE A NAME TO THE FILE
       UNKNOWN FILE #00
```

Obviously, FIXCAT has no idea what the name of the file should be. It is prompting you with a name which is probably unique. If, after examining the sector dump, you think you know the name of the file, you can change this prompt and put the actual name into the catalog at this time. We don't know which file this is so we must accept the prompt of UNKNOWN FILE #00.

Now FIXCAT will display some more information about the file:

```
FILE: UNKNOWN FILE #00

TRACK/SECTOR LIST: T$12 S$0F
DATA: T$12 S$0E

TOTAL SECTORS ALLOCATED TO FILE: $0002
```

The location of the track/sector list (TSL) for the file is given as track $12, sector $0F. The file contains only one data sector, at track $12, sector $0E, for a total of 2 sectors (counting the TSL itself). Now FIXCAT asks for the type of file:

```
WHAT TYPE OF FILE IS IT? (T,I,A,B,R,S)
 A
```

Notice that FIXCAT has guessed that the file is an APPLESOFT program. It turns out that this guess is correct, since this file is, in reality, the HELLO program for the diskette! Of course you don't know this so you can only hope that FIXCAT has guessed correctly and therefore accept its prompt of A.

This process now repeats itself for every file on the diskette. Other files are located, their first sector is displayed, and you accept all of the prompts to recover them. When the scan completes you will see this message:

```
 72 SECTORS IN USE
488 SECTORS FREE

ONE OR MORE ERRORS WERE FOUND IN THE
VTOC BIT MAP.  CORRECT THEM?
 Y
```

This message indicates that the computed freespace map does not match the one found on the diskette. Obviously it doesn't since the one on the diskette is nothing but zeros! Reply Y to cause FIXCAT to substitute the computed freespace map for the zeroed one. Now you will see:

```
PROCESSING COMPLETED.

APPLY ACCUMULATED CORRECTIONS TO THE
VTOC/CATALOG TRACK?
```

Reply Y to this question to cause FIXCAT to write back its reconstructed catalog track to the practice diskette. FIXCAT will now terminate. Boot DOS and CATALOG the practice diskette. You will see this:

```
DISK VOLUME 001

A 002 UNKNOWN FILE #00
B 002 UNKNOWN FILE #01
A 002 UNKNOWN FILE #02
T 002 UNKNOWN FILE #03
```

Now it is time to try to identify the files and rename them to something more sensible. LOAD the first file into BASIC and LIST it. It is the HELLO file (10 END) so RENAME it as follows:

```
RENAME UNKNOWN FILE #00,HELLO
```

Since there was only one binary file on the practice diskette, it is an easy task to identify it:

```
RENAME UNKNOWN FILE #01,BINARY FILE
```

LOAD the other APPLESOFT program and LIST it. It is the program we called APPLESOFT PROGRAM:

```
RENAME UNKNOWN FILE #02,APPLESOFT PROGRAM
```

This leaves only the T (text) file called TEXT FILE:

```
RENAME UNKNOWN FILE #03,TEXT FILE
```

This diskette is back to exactly the same state prior to our zeroing its catalog track.

Of course, identification of files on other diskettes may not be as easy as this. Binary files can be BLOADed and examined in memory (or with ZAP). Text files can be examined with ZAP, and BASIC programs can be LOADed and listed. FIXCAT will occasionally guess the file type incorrectly. If this happens, DELETE the file and run FIXCAT again, searching for lost files. When the file is found again, override the prompt on the file type and try something else. In general, FIXCAT's algorithm for determining file types will be accurate about 95% of the time. It is never accurate for S or R files, however. Anyway, the important thing is that the files are recovered!

APPENDIX
A DISCUSSION OF DISK I/O ERRORS

The purpose of this appendix is to provide information to the Apple disk user that will be of some help in both understanding and correcting diskette errors. This discussion is by no means exhaustive nor highly technical. However, it does summarize some of the insight the authors have gained through many hours spent attempting to repair diskettes, in many cases successfully.

While we know that you have heard this particular message before, it bears repeating. **Back up your important diskettes!** We have learned by experience that no matter how careful you are, sooner or later you will encounter an I/O error on a diskette. You probably have some diskettes that are not recoverable by any practical means besides backup, and these are the most important diskettes to have copies of. In this case, we recommend heeding the warning of the pessimists who maintain that your most vital data will always fail first.

SYMPTOMS AND POSSIBLE SOLUTIONS

Below are listed some of the symptoms you are likely to encounter if a diskette has been damaged in some way. In each case the likely location of the error is given and a possible solution is suggested. The section titled Data Errors contains more detailed information on data recovery and reconstruction.

Disk doesn't boot (spins with no recalibrations)

There are two possibilites. First, you may have a good diskette, but you are trying to boot a 13 sector diskette without using the BASICS diskette. Second, track 0 of the diskette is damaged. Most likely at fault is sector 0 of track 0. Try putting a new copy of DOS on the diskette.

Disk doesn't boot (recalibrates)

It is likely that the portion of the diskette containing DOS (Tracks 0-2) is damaged. Install a new copy of DOS.

Disk starts to boot then recalibrates and I/O error message displayed

The VTOC is read first, then the CATALOG sectors, and finally the "HELLO" file, any one of which could cause the error. To further isolate the area try to CATALOG the diskette or LOAD the "HELLO" program.

I/O Error on CATALOG command

Either the VTOC or one or more CATALOG sectors are damaged. FIXCAT should provide a solution to this problem.

I/O Error during file access (LOAD, BLOAD or READ)

Either the TRACK SECTOR LIST or one or more sectors in the file itself are damaged. ZAP should help you further isolate the exact location.

Recalibration occurs during successful operation

While the diskette may simply be off center, it is likely that an error occurred from which DOS was able to recover by repeatedly attempting to read the sector in question. This type of error may become permanent and should be fixed quickly. Copy the diskette if possible, and see Data Errors for information on rewriting the faulty sector(s).

NATURE OF DISKETTE ERRORS

Media Errors

A media error means that there has been some physical damage to the media itself. You usually will not be able to reformat a diskette with a media error, although some utilities exist that simply mark the damaged area in use, allowing the rest of the diskette to be used. It would not be advisable to run FIXCAT on a diskette that has a suspected media error.

Data Errors

A data error may or may not be recoverable, but the media itself is undamaged and can be reformatted. There are two categories of data errors.

The first category of data errors includes those that are caused by airborne contaminants, random electrical noise, or small magnetic defects not detected during the write operation. These errors tend to be recoverable because a good read of the data can often be achieved by repeated attempts to read the damaged sector. The first step should be to reread the sector 12 times or until a good read is accomplished. If that fails, read adjacent tracks and then attempt to reread the damaged sector. It might also be wise to remove the diskette, center the media in its jacket, and reinsert it in the disk drive. If the data cannot be recovered by the above mentioned technique, it is unlikely to be recoverable through normal means. Because there are many different ways a particular error can occur, it is beyond the scope of this manual to deal with all possibilities. On rare occasions it is possible to determine the precise abnormality using TRAX, and either patch DOS or write a special program to read a sector with that particular abnormality. This is only suggested for users with an excellent understanding of DOS and diskette data formatting.

The second category of data errors consists of those errors that always destroy data. There are two ways that these errors manifest themselves, and it is suspected that hardware is the cause in both cases. The first type of error in this category is a sector that can be read but now contains spurious data. The error occurs while the data is in memory and then the bad data is written to the diskette. The classic example is to load your BASIC program into memory only to discover some portion of it is now garbage. Unless you have another copy of that particular portion of data, it is lost. The second error occurs while writing to the diskette and totally destroys one or more sectors, in some cases an entire track. An example would be to SAVE a file to disk and then discover that the VTOC and one CATALOG sector was destroyed. That data, although not readable, may not necessarily be lost because those areas of a diskette are usually reconstructable. What follows is a brief description of how to deal with particular kinds of data loss.

We can break the diskette down into six basic kinds of data storage, which are as follows:

DOS
VTOC
CATALOG
TRACK SECTOR LIST
FILE
UNUSED SECTORS

The first three are all reconstructable, although some information, namely program names, may be lost. Fortunately, that is often not critical. DOS is restored by running the appropriate utility on the System Master diskette (MASTER CREATE on DOS 3.3) and the VTOC and CATALOG can be reconstructed by FIXCAT. Either situation might call for the use of INIT first if you are unable to write to the diskette. (See the tutorial entitled FINDING AND CORRECTING FORMAT ERRORS.)

Of the last three, when the error occurs in an unused sector, it poses little threat (although it would be best to deal with it to insure no future problems). INIT works very well to recover those sectors. The remaining two, TRACK SECTOR LISTS and FILES, are often impossible to reconstruct. The degree of difficulty depends entirely on the particular diskette (how many files it contained, etc.) and on your knowledge concerning the missing data. It might be easy to reconstruct a TRACK SECTOR LIST on a relatively empty diskette, but the task may prove impossible on a full diskette with many files. FIXCAT can provide you with a great deal of helpful information by locating all valid TRACK SECTOR LISTS and FILES. If the error is in a FILE you are really at the mercy of how much you know about the FILE. See the tutorial titled LOCATING AND FIXING AN I/O ERROR for some hints on how to proceed.

BACKING UP DAMAGED DISKETTES

When trying to recover data from a damaged diskette, it is often best to make a copy of the diskette to avoid further loss of data. The problem is that many copy programs will not copy a diskette with one or more bad sectors. There is a relatively easy patch that allows the copy program provided on the Apple 3.3 System Master to ignore I/O errors. The patch forces the program to continue until all 35 tracks have been attempted, recalibrating two times for each sector not found. The following paragraph explains how this patch can be made.

Although the patched copy program will continue to operate correctly for normal diskettes, you should make a copy of your System Master before proceeding. To apply the patch, boot the Bag of Tricks diskette and select ZAP. When ZAP is running, insert the copy you made of the System Master into either disk drive. (If drive 2 is used select it by typing S,2 on the ZAP prompt line and press the RETURN key.)

Now type the ZAP command OPEN'COPY.OBJ0' followed by the RETURN key. This will display the first portion of the System Master copy program on the screen. Type E5 to position the cursor to the correct byte. The inverse cursor should display the value 60. If this is not the case, please check and see that you have followed directions correctly. The value 60 needs to be changed to an EA. Simply type :EA followed by a RETURN to make the change. After verifying that you have made the patch correctly, type UNLOCK WRITE and press RETURN. Your patch is now complete. The patched System Master copy can be used to copy damaged diskettes.

NOTES

Bag of Tricks

QUALITY SOFTWARE

19810