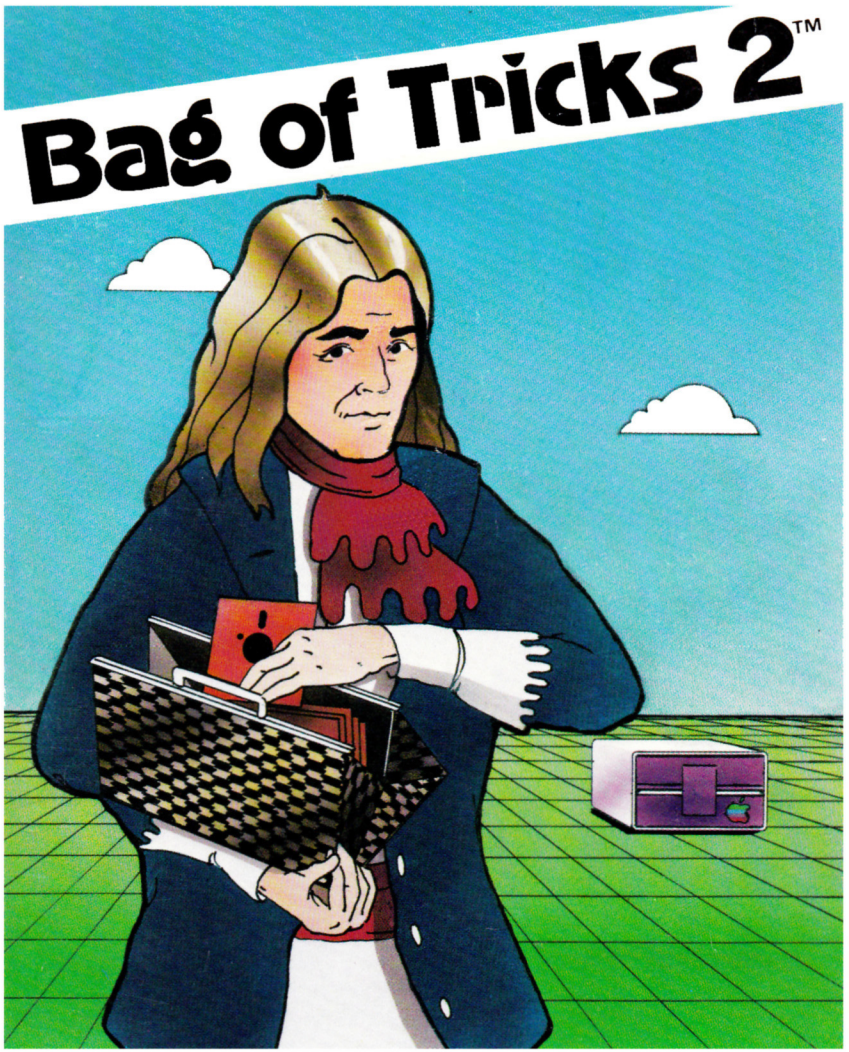
NEW!

Bag of Tricks 2™



FOUR DISK UTILITY PROGRAMS
FOR USERS OF APPLE II, II+, IIe AND IIc

BY THE AUTHORS OF *Beneath Apple DOS*
AND *Beneath Apple ProDOS*

Don Worth and Pieter Lechner

QS QUALITY SOFTWARE

Bag of Tricks 2™

by Don Worth and Pieter Lechner

QS QUALITY SOFTWARE
Computer Book Division
21601 Marilla Street, Chatsworth, CA 91311
(818) 709-1721

International Standard Book Number: 0-912985-35-6

87 86 85 5 4 3 2 1

Printed in the United States of America

TABLE OF CONTENTS

TABLE OF CONTENTS

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 2 is a collection of four utility programs. Together these programs can accomplish most of the functions Apple II disk owners need for manipulating data on disks as well as error detection and recovery.

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. Bag of Tricks 2 adds a few new features that users have requested, but primarily it expands the original Bag of Tricks so that many of its functions can work not only with 5 1/4" floppies, but also with other popular disk devices such as hard disks, 3 1/2" floppy disks, and RAM disks.

As a result, Bag of Tricks 2 remains unique in that it can perform more useful functions than any other disk utility available for the Apple II computer. Hopefully 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 2 DISKETTE

Bag of Tricks 2 consists of four separate programs provided on a standard ProDOS formatted 5 1/4" diskette. The best way to boot Bag of Tricks 2 is to turn off the computer, place the diskette the drive you normally boot from, and turn on the computer. Bag of Tricks 2 will load itself in and display the main menu screen, which asks you to press a key to select one of its four programs. It is also possible to boot the program by inserting the diskette in the proper drive and typing PR#6 [RETURN] (or other appropriate slot number).

Bag of Tricks 2 will load and operate on any Apple IIe or Apple IIc. It will also operate on an Apple II Plus if the computer is configured for 64K of memory. All programs except ZAP use only the 40-column text mode. ZAP can operate in either a 40-column mode or, if the capability exists, in an 80-column text mode. Apple II Plus owners with a non-Apple 80-column card will have problems running ZAP in 80-column mode unless it is modified for their installation. See the tutorial in Chapter 6, "Modifying ZAP to Work with a Videx 80-Column Card".

Those who understand the ProDOS operating system will notice that the "main menu" program for Bag of Tricks 2 is a file called "BOT.SYSTEM". RUNning this file with ProDOS active will run the Bag of Tricks program. This method of booting Bag of Tricks 2 is not recommended, however, because the version of PRODOS on Bag of Tricks 2 is the version it expects to work with. This is version 1.1.1 with a modification to the Disk II Device Driver to allow up to 50 tracks on Disk II devices. If this version of PRODOS is not active, there may be undesired results.

Except for INIT, the Bag of Tricks 2 programs can support an optional printer. The printer interface card can be in any slot.

Bag of Tricks 2 supports the standard ProDOS QUITCODE. If you wish to continue on to another SYSTEM program (such as BASIC.SYSTEM) after using Bag of Tricks 2, press Q when the main menu screen is visible.

The Bag of Tricks 2 diskette may be backed up, and we recommend that you make a backup copy for your use and keep the original in a safe place. We also recommend that you become a registered owner of Bag of Tricks 2 (see the last section of this chapter).

TRANSFERRING BAG OF TRICKS TO ANOTHER PRODOS VOLUME

Many users will want to put Bag of Tricks on a 3 1/2" floppy or on a hard disk. Certain precautions should be used when doing so. First of all, all of the files must be in the Volume Directory. The Volume Directory can have any name. The four program files ZAP, TRAX, INIT, and FIXCAT cannot run by themselves. The files BOT.SYSTEM, PRODOS, BOT2PIX, and ZDOS are needed to run any of the four programs. FIXCAT and INIT also require the file MENU.

As mentioned above, the version of PRODOS on the Bag of Tricks 2 diskette has been slightly modified. If your Volume Directory already has a version of PRODOS on it, you must decide whether or not to replace it with the Bag of Tricks 2 version of PRODOS. Replacing it will allow Bag of Tricks 2 to operate properly with floppy disks of more than 35 tracks, but replacement may not be best for other software on the volume.

THE FOUR PROGRAMS AND WHAT THEY DO

Although the four programs that comprise Bag of Tricks 2 can often be used together to perform some task, the purpose of each is quite

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

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 5 1/4" 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.

INIT:
The INIT program can be used to reformat a single track on a 5 1/4" 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. INIT will also allow you to specify the order of the sectors on any given track. INIT also provides a copy program that allows you to copy a damaged disk, restoring all the good data and reformatting any damaged sectors.

ZAP:
ZAP in its simplest sense allows you to read and modify a disk at the block or track/sector level. Disk data 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, ProDOS files, CP/M files, and PASCAL files. ZAP is in some ways the most complex of the four programs, but for most users it is the most frequently used.

FIXCAT:
The FIXCAT program is an automated utility which allows you to diagnose and correct errors in the catalog track of a DOS diskette or the directory files of a ProDOS diskette. This is a critical utility because the catalog or directory are normally the most frequently read and written to areas on the disk. Thus they are the most likely to be damaged, and when they are damaged you lose access to all your files! FIXCAT has recovered hundreds of lost disks over years, and in many cases it has done so **automatically**, with no significant user input!

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 2. In the next four chapters each of the programs is covered separately. The reference material you will want to refer to frequently is in the front of each of these chapters. At the end of each chapter you will find a functional description of the program and a tutorial. The last chapter (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 2 but also should be of value in fixing diskettes or patching DOS, ProDOS, and other programs. We suggest that you first read over the functional 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.

BECOMING A REGISTERED OWNER

We strongly recommend that you fill out and return the Bag of Tricks 2 registration card that came with your copy of Bag of Tricks 2. Only registered owners will be eligible for discounts on future upgrades, if any, to Bag of Tricks. Registered owners will also be placed on Quality Software's mailing list, which means they will receive all of Quality Software's mailings announcing availability of Apple-related products.

CHAPTER 2

TRAX—By Pieter Lechner

We have put TRAX first in the order of the Bag of Tricks 2 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.

IMPORTANT NOTE: Unlike the other Bag of Tricks 2 programs, TRAX will not operate with any disk device other than a Disk II or equivalent 5 1/4" floppy disk drive. When you run TRAX, it looks for a Disk II drive. If it finds none, it immediately exits.

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 5 1/4" floppy diskette, which is useful in better understanding DOS. While TRAX is designed to operate on normal (unprotected) 13- or 16-sector diskettes, its data reading technique allows it to read any 5 1/4" floppy diskette created using a Disk II compatible disk controller.

If you have never used TRAX before, you will want to skip over the reference materials at the beginning of this chapter and read the sections titled "TRAX--A Functional Description" and "A TRAX Tutorial." TRAX documentation is organized as follows:

ALPHABETICAL LISTING OF TRAX 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.

TRAX COMMAND DESCRIPTIONS

Miscellaneous Commands are valid in either mode. Analysis Mode Commands are valid only in the Analysis Mode. Raw Data Mode Commands are valid only in the Raw Data Mode.

MISCELLANEOUS COMMANDS

?[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 TRAX expression, which consists of one or more numeric terms connected by addition or subtraction operators.

END

The END command exits TRAX and returns to the Bag of Tricks 2 main menu screen.

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 the PRINT command. The printer slot in use is displayed on the top line of the display.

PRINT

Copies the entire screen image to your printer. This comand may be issued in either the analysis or raw data mode.

ANALYSIS MODE COMMANDS

0

Force recalibration of the disk arm followed by a read, analysis, and display of track 0. Use this command if TRAX seems to be reading the wrong track.

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, use the C command to return the screen to its normal display.

C

Display the normal analysis screen, which shows in the center column the actual address checksums read from the disk. This command is usually used to clear the display after an A or D command has been issued.

D

Display the result of the checksum computations for the **data** fields of each sector. The results are shown in the center 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, use the C command to return the screen to its normal display.

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.

N ..or.. N[EXP]

Read the next track (i.e. the current track plus one), analyze the data, and display the results. If [EXP] is present, read the track that is the current track plus [EXP].

P ..or.. P[EXP]

Read the previous track (i.e. the current track less one), analyze the data, and display the results. If [EXP] is present, read the track that is the current track less [EXP].

R ..or.. R[EXP]

If the operand [EXP] is omitted, then read the track indicated by the track number displayed at the top of the screen, analyze the data, and display the results. If [EXP] is specified then read the track indicated by [EXP].

S[EXP1],[EXP2] ..or.. S[EXP1] ..or.. S,[EXP2]

The S command sets the slot and/or drive for subsequent disk reads. The slot and drive number are initially set to those used to boot Bag of Tricks 2, or to the highest-numbered slot with a 5 1/4" floppy drive, if Bag of Tricks 2 was not booted from a 5 1/4" floppy. If you want to change them you may use this commmand. The first expression, [EXP1], is the slot number and may be relative or absolute, ranging in value from 0 to 7. If you specify a slot that does not have a 5 1/4" floppy disk drive connected, the error message "NOT A FLOPPY" will result. [EXP2] is the drive number, 1 or 2, and may be relative or absolute. If [EXP1] is omitted, the current slot remains unchanged. If [EXP2] is omitted, drive 1 is assumed.

V

Verify the diskette starting at the next track and continuing through track 49 (Hex 31). Any abnormalities are displayed, stopping the verification process. The verification process may be continued by pressing V again. Verification will normally stop when the last track on the disk has been read and TRAX is attempting to verify a non-existent track.

X

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

RAW DATA COMMANDS

[EXP]

Scroll forward [EXP] lines (one line is eight hex bytes) in the hex display of the track. If [EXP] is negative, then scroll back. [EXP] can range from -9 to +9.

B

Go to the beginning of the raw data buffer.

E

Go to the end of the raw data buffer.

G ..or.. G[EXP]

Go to the current buffer address. If [EXP] is present, set the current buffer address to EXP and go to it. The buffer address must be in the range $0 to $1AFF.

L ..or.. L[EXP]

Look for the current search byte. If [EXP] is present, then change the current search byte to [EXP] and look for it. The search always starts with the second line on the screen. 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.

N

Scroll forward one page (80 hex bytes) in the raw data buffer.

P

Scroll backward one page (80 hex bytes) in the raw data buffer.

X

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

TRAX ERROR MESSAGES

It should be pointed out that although TRAX is accurate in most cases, some unforseen 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.

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.

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

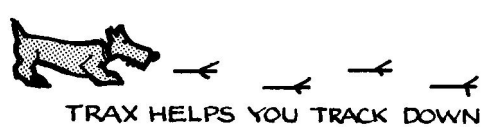
This message will probably occur if Format is set to 13 and a 16- sector disk is used. In the case of certain damaged diskettes, TRAX might not be able to distinguish between 13- and 16-sector formats.

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.

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.



END OF BUFFER

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

MISSING OPERAND

The command you have used is a legal one but requires at least one operand.

NO DATA

When displayed on the command line, this indicates that the command just tried can only be executed if data has been read in. There is either no data in the data buffer or a Slot command has been issued, effectively clearing the buffer. Issue a command such as 0 or R that will read data into the buffer, then try the command again.

The "NO DATA" message may also appear on the analysis display. This will generally happen only with 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 blank 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.

NOT A FLOPPY

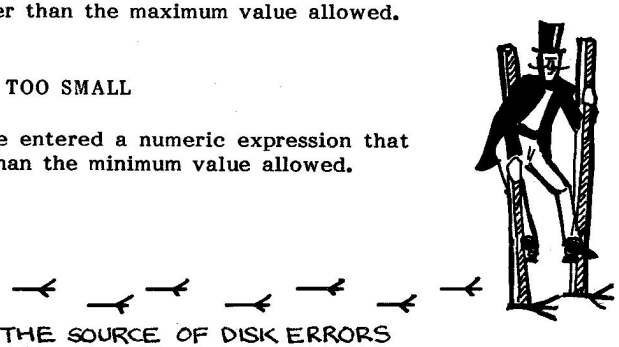
You have tried to access a slot that does not have a controller card recognizable as a 5 1/4" floppy disk drive.

NUMBER TOO BIG

You have entered a numeric expression that is greater than the maximum value allowed.

NUMBER TOO SMALL

You have entered a numeric expression that is less than the minimum value allowed.



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.

SYNTAX ERROR

You have typed a non-existent command or the command operand is not in the right format. Check the command line.

TOP OF BUFFER

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

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 controller ROM.

VERIFICATION COMPLETE

This message indicates the successful end of a Verify command. It should only occur if you have a 50-track disk drive and a diskette with all 50 tracks formatted. We've never heard of such a drive, so we don't expect this message. Normally the V command will end when it tries to read a non-existent track (such as track $23 on 35-track drives or track $28 on 40-track drives).

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.

WRONG MODE

You have entered a command that exists but can only be used in the other operating mode. For example, L is a raw data mode command and trying to use the L command in analysis mode will result in this message.

TRAX—A FUNCTIONAL DESCRIPTION

This section describes how to use TRAX. It is intended for first time users and for those who have not used TRAX for a while. This section also includes some discussion of the problems of analyzing raw disk data.

If you are unfamiliar with TRAX, it is recommended that you read over this section quickly, then perform "A TRAX Tutorial" (the following section), and return to this section to study it in more detail.

Once you become familiar with TRAX, you should find the reference material in the front of the chapter sufficient to operate the program.

Throughout this chapter we 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 **Beneath Apple DOS**, or Appendix C of **Beneath Apple ProDOS.** These contain fairly complete discussions of Apple II 5 1/4" diskette formatting.

TRAX COMMAND SYNTAX

In general, TRAX commands are similar in syntax and function to ZAP commands. Command names consist of one or more characters and may require zero, one, or two operands. No spaces may appear between the command name and its operands. Only one command may be issued at one time using TRAX.

A nice feature of TRAX allows you to re-enter the previous command with a single keystroke (RETURN is not required). If you have not pressed any keys since the last command was entered, the TAB key (CTRL-I on Apple II Plus) automatically reenters and executes the previous command.

If an error occurs during the execution of any command, processing stops and an error message is displayed. Error messages are described in detail in a previous section of this chapter.

TRAX commands can be divided according to operating modes. Some commands are available in either operating mode, some are only available in the analysis mode, and some are only available in the raw data mode.

MISCELLANEOUS OPERATIONS

Related Commands:

END	EXIT PROGRAM TO MAIN MENU
PR#[SLOT]	SET THE PRINTER SLOT
PRINT	DUMP SCREEN TO PRINTER
?	CALCULATOR COMMAND

These four commands are available in and have the same function in both the analysis mode and the raw data mode.

ANALYSIS MODE

Related Commands:

0	RECALIBRATE AND READ TRACK 0
A	SHOW ADDRESS FIELD CHECKSUM
C	DISPLAY ADDR CHECKSUM VALUES
D	SHOW DATA FIELD CHECKSUMS
F	TOGGLE FORMAT, 13- OR 16-SECTOR
N[EXP]	READ NEXT TRACK PLUS EXP
P[EXP]	READ PREVIOUS TRACK MINUS EXP
R[TRACK]	READ A TRACK
S[SLOT],[DRIVE]	SET SLOT AND DRIVE
V	VERIFY FROM NEXT TRACK ON
X	GO TO RAW DATA MODE

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.

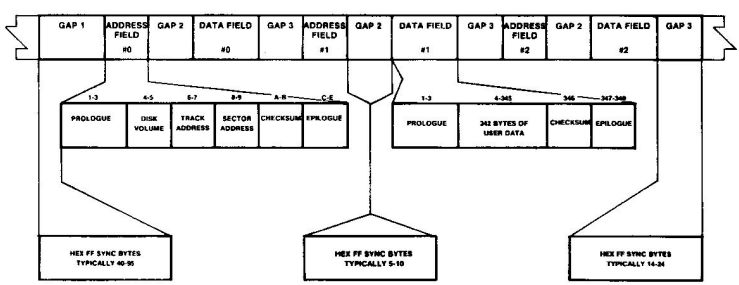
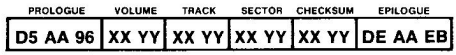


Figure 2.1 Track Format



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

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.

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.

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.

DATA FIELD

SIX AND TWO ENCODED

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 the analysis mode will not correctly diagnose every type of I/O error, it will uncover formatting errors a high percentage of the time.

Another fact to remember is that the sector number displayed on the analysis mode display is the number of the sector as recorded in the address field. **This number is not necessarily the same as the logical sector number used by the operating system.** The various operating systems available for the Apple II use a table to translate the number in the address field to the logical sector number. This is discussed in more detail in Chapter 3.

RAW DATA MODE

Related Commands:

[EXP]	MOVE FORWARD (BACK) EXP LINES
B	GO TO BEGINNING OF BUFFER
E	GO TO END OF BUFFER
G[OFFSET]	GO TO BUFFER OFFSET
L[BYTE]	LOOK FOR SEARCH BYTE
N	SCROLL TO NEXT PAGE
P	SCROLL TO PREVIOUS PAGE
X	RETURN TO ANALYSIS MODE

A brief explanation of how TRAX reads raw data is in order. Raw data bytes (disk bytes) are 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 "beginning" of the track (the end of gap 1 of Figure 2.1) is then placed at the beginning of the raw data buffer. The raw data buffer is $1B00 bytes long, more than enough to hold the disk bytes on a 5 1/4" floppy disk track, which is typically about $1900 bytes long.

Note that all Apple disk bytes have the high bit on. Chapter 3 of **Beneath Apple DOS** and Appendix C of **Beneath Apple ProDOS** explain how disk bytes relate to data bytes.

A TRAX TUTORIAL

The following tutorial is provided to familiarize the first-time user with the features and operation of TRAX. Boot the Bag of Tricks 2 diskette (see Chapter 1 for loading instructions), press T to select TRAX, and follow along with the tutorial.

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, type R and press the RETURN key. 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 look like Figure 2.4.

TRAX is now in the Analysis Mode. Most of the vital information used by the disk operating system to locate and read data is displayed in this mode. It is important that this information is correct, because if a single byte is altered the entire sector is unreadable. 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 examining the Bag of Tricks 2 diskette, which is a normal, hopefully 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 type A and press the RETURN 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.

Now enter the D command. 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. Enter the C command to restore the screen to normal. For a detailed explanation of the construction of address and data fields, see **Beneath Apple DOS** (Chapter 3).

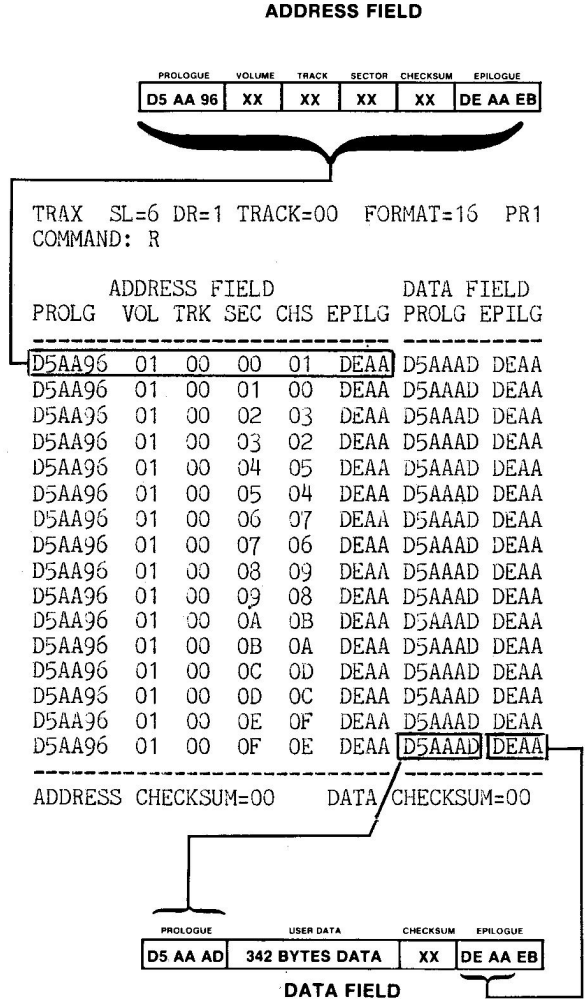


Figure 2.4 Analysis Mode Display

Now enter the X command. This puts you in the Raw Data Mode, where the raw disk data on the track is displayed. Figure 2.5 is a typical raw data display, and points out how address fields and data fields can be identified in the raw data.

While the raw track image is difficult to decode manually, it is nevertheless informative and useful for viewing tracks that TRAX was unable to interpret. TRAX stores the track image in a buffer in memory. Buffer addresses range from $0000 to $1AFF. 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 raw data buffer. Type 1 and press the RETURN key. This scrolls the display up one line. Now type 9 and press RETURN. Numbers from -9 to +9 will move the display ahead or back the appropriate number of lines. Sixteen lines (128 disk bytes) are displayed on the raw data display. Enter the N (Next) command and the screen will scroll one page (16 lines) forward. The opposite command P (Previous) scrolls back one page.

You can move to the beginning of the buffer by using the B command and to the end of the buffer by using the E command. You can go to a specific spot in the buffer with the G (Goto) command. Enter G1000. Then issue an E command. Then simply enter G. The program remembers the last Goto buffer position.

It is impossible to scroll past either the start or end of the buffer, and any attempt to do so will result in an error message.
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.

You can also search for a particular byte when in the Raw Data Mode. This is done with the L command. At this point the top of the display identifies the current search byte as D5. This is the default value, because D5 is the byte whieh normally begins all address and data fields. Each time you issue the L command the next D5 in the raw data buffer 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. Enter the command LAD (Look for $AD). This changes the search byte to $AD and looks for the next occurrence of $AD in the buffer. Parentheses will be placed around the value of the current search byte. Experiment with other values for search byte, then exit the Raw Data Mode by pressing the X key. You will be returned to the Analysis Mode.

When you are using TRAX to analyze diskettes, you should keep in mind that the sector order in the Analysis Mode display is the same as the order of the data in the Raw Data Mode 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 back and forth between the two modes.

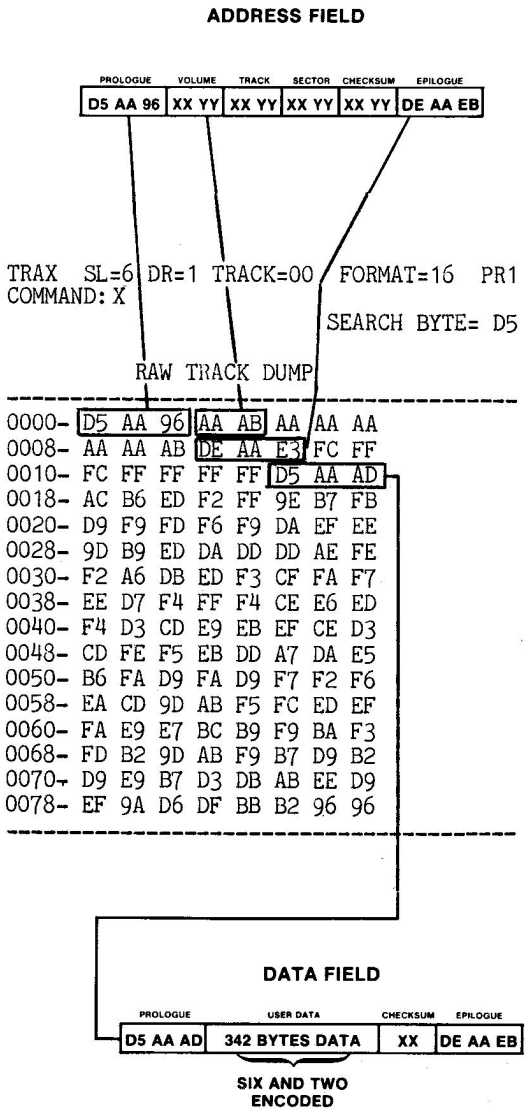


Figure 2.5 Raw Data Mode Display

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 3 as the **physical** sector numbers. The **logical** sector numbers used by the various operating systems are usually different than the physical sector number.

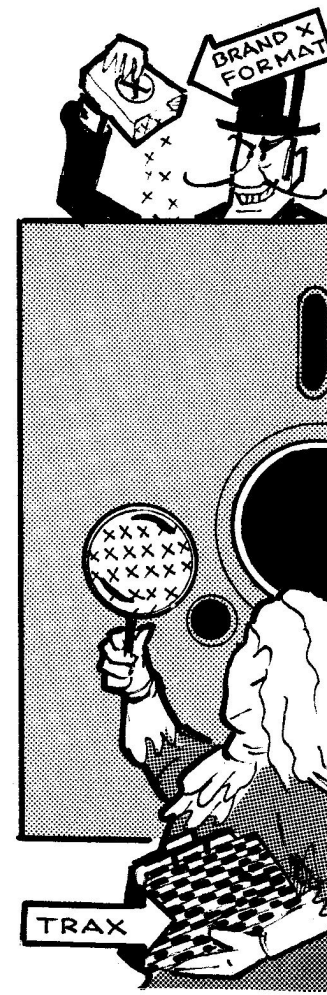
When you are in the Analysis Mode, you can change tracks. Enter the N command. This moves you to the Next track, reads that track, and does an analysis of it. When this processing is complete, you will see that the track number is now 01.

To access the Previous track, enter the P command. Now you are back to track 0.

To select a particular track, use the R command with an operand. For example, read track 11 (Hex) with the command R11. Track $11 is read and analyzed, and the results of the analysis are displayed. 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.

Enter the 0 command. This will recalibrate the disk arm and read track 00. Now simply press V to verify the diskette. TRAX will verify that all sectors are readable,



TRAX ALLOWS YOU TO INSPECT DISKETTES FOR NON-STANDARD FORMATTING

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 until you come to track $23. There is no such track on the Bag of Tricks 2 diskette, which is only formatted for 35 tracks.

This ends our TRAX tutorial. If you have any damaged or copy-protected diskettes, you may wish to look at them to see the kind of results that occur when disks are not normal. TRAX always reads the diskette, and never writes to it.

CHAPTER 3

INIT—By Pieter Lechner

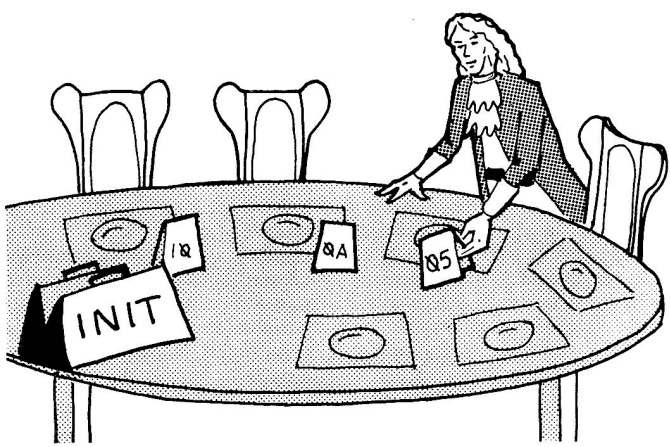
The INIT utility has two parts, an INIT function and a COPY function. The INIT function allows you to format a range of tracks on a 5 1/4" floppy diskette--from a single track to an entire diskette. Both 13- and 16-sector formats are supported. Good sectors on a track can be preserved, and all bytes in bad sectors are set to zero. This allows you to reformat a damaged track without removing your diskette from the disk drive.

The COPY function of INIT allows you to copy a damaged disk volume. This function will work for any disk storage device that allows block reads (any device ProDOS will run on). This option automatically saves all readable parts of the disk volume. Unreadable blocks or sectors on the original disk will be readable on the copy, but the data in those blocks or sectors will be garbage. This feature, used in conjunction with ZAP or FIXCAT, is very helpful in recovering damaged disks.

Another function of INIT is to reorder sectors on each track of a 5 1/4" floppy diskette. This subject is covered in detail later in this chapter, in the section "How Sector Skewing Can Affect Disk Performance." By selecting the appropriate skewing you can optimize the speed with which the operating system 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.

INIT is self-prompting and to some extent self-explanatory. If you have never used INIT before, however, you should read the following three sections on INIT parameters, copying disks, and error messages. Then perform the tutorial at the end of the chapter. INIT documentation is organized as follows:

ALPHABETIC LISTING OF INIT PARAMETERS



SPECIFYING SECTOR ORDER

DESCRIPTION OF INIT PARAMETERS

Using INIT and setting INIT parameters is accomplished by following these general rules:

Yes/No selections:

Y,y, or left arrow	Select yes
N,n, or right arrow	Select no
RETURN	Accept current selection
ESC	Go backward in program

Menu selections:

right or down arrow	Move down (or to top if at bottom)
left or up arrow	Move up (or to bottom if at top)
Number	Move to the numbered selection
RETURN	Accept current selection
ESC	Go backward in program

Input a parameter--first keystroke:
(the cursor is displayed to the right of the default value)

RETURN	Accept default
left arrow	Move cursor into default, allow editing
right arrow	Begin editing at end of default value
CTRL-D, DELETE	Edit default value, delete last character
ESC	Go backward in program
Any other key	Clear default, begin editing new value

Input a parameter--subsequent keystrokes:

RETURN	Accept entire input field
left arrow	Move cursor left within input field
right arrow	Move cursor right within input field
CTRL-D, DELETE	Delete character to the left of cursor
ESC	Cancel input, go backward in program
Any other key	Add character to input field

If you select the **INIT** option from the first INIT menu, you will be allowed to set the following parameters:

SECTORS PER TRACK

This value is the number of sectors per track. Enter 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).

OPERATING SYSTEM

This is the type of operating system being used on this diskette. Enter D for DOS, P for ProDOS or Pascal, or C for CP/M. For 13 sectors both DOS and CP/M are available. For 16 sector diskettes DOS, CP/M and ProDOS/Pascal are available.

PRESERVE DATA

This question asks whether the data currently on your diskette should be preserved. Enter 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, and then the track is reread and verified against the data. 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 can not be reformatted. For this reason, we recommend that you make a backup before using INIT to preserve data. The COPY option at the beginning of INIT may be used to make the backup copy. 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. Enter A for ascending or D for descending. Ordinarily, DOS reads sectors in DESCENDING order while all other operating systems read them in ASCENDING order.

OPTIMAL SKEWS FOR LOADing

Operating System	Skew Direction	Skew Factor
DOS 3.2	DESCENDING	6
DOS 3.3	DESCENDING	9
ProDOS	ASCENDING	2
Pascal	ASCENDING	2
CP/M	ASCENDING	3

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 controller card. Slots 1, 2, 3, 4, 5, 6, and 7 are supported, but only if a controller card is installed there.

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. Enter D for Default or any number from 0 to 254 ($00-$FE Hex). The DEFAULT option 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 0-49 ($00-$31 Hex).

ENDING TRACK

The last track to be formatted. The options are 0-49 ($00-$31 Hex). This value must be greater than or equal to the value for STARTING TRACK. The default value is 34, the highest track number of a standard Disk II diskette.

COPYING DISKS

The **COPY** option, selected from the first INIT menu, allows you to copy a disk volume. This copy routine works with any disk device that ProDOS can operate with (the disk that is copied does not have to be a ProDOS disk). You may copy from and to the same drive, or you may copy from one drive to another drive. In the latter case, the drive you are copying to must be expecting a volume that is exactly the same size as the volume in the drive you are copying from.

The disk that you are copying to **must be formatted** before selecting the COPY option. If you are copying to a 5 1/4" floppy diskette, the formatting can be performed by INIT using the "INIT a disk" option. Any other type of diskette must be previously formatted in the normal manner using some software other than Bag of Tricks 2.

If you select the COPY option from the first INIT menu, you will be guided by menus to select the source and destination drives that you wish to copy from and to, respectively. All the drives that are connected to your system will be listed for you to select from.

If you are copying from and to the same drive, you will be prompted when to put the source and destination diskettes into the drive. Two-drive copies will be performed automatically.

Whenever the INIT COPY routine encounters an error while reading or writing, a message will be printed on the video screen. The block number the program was trying to read or write and the error number (ProDOS MLI error number) will be displayed.

You may abort the COPY process at any time by pressing the ESC key.

Because the primary purpose of the INIT COPY routine is to copy damaged diskettes, the copy process will not stop when errors are encountered. If a block on the disk being copied cannot be read, **that block on the copy will have "garbage" data in it.** These "garbage" blocks will be readable by ZAP, so that repairs can be attempted. If the damaged diskette is a 5 1/4" diskette, then analyzing the original with TRAX will identify which sectors are damaged and therefore contain garbage on the copy. For other disk devices, use ZAP and try to read all the blocks on the original. An I/O error will occur when ZAP tries to read the unreadable blocks.

INIT ERROR MESSAGES

IS THE ABOVE OK ? <YES> NO

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.

EXISTING DATA WILL BE OVERWRITTEN
INSERT DISK IN DRIVE XX SLOT XX

This message 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 DISK IN DRIVE XX SLOT XX

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

When not preserving data, this message indicates the track currently being formatted.

BUILDING CATALOG 17-DEC 11-HEX

This message occurs only when formatting track 17 ($11 Hex), not preserving data, and using DOS format. Both an empty catalog and a VTOC (Volume Table of Contents) are constructed at this time. INIT assumes that the diskette contains no DOS boot image, and the new VTOC is marked to indicate this.

READING TRACK XX-DEC XX-HEX

This message occurs when preserving data and data is being read 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 preserving data and data is being written 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 * *

This message indicates that the write protect switch on your disk drive is closed. Most likely this means that the notch in your diskette has been covered. You must remove the write protect tab from your diskette if you wish to write to it, then press RETURN to proceed. Press escape if you don't wish to write.

* * XX OF XX SECTORS UNREADABLE * *

This message only occurs when you are preserving data. It tells you how many sectors could not be preserved on the indicated track. If you press

RETURN to proceed, the unreadable sectors will be replaced with zeroed sectors on the newly formatted diskette. Escape will abort the reformat and write. If all sectors on the track are unreadable, you may have an unformatted diskette in the drive, or there may be no diskette at all in the drive.

* * UNABLE TO FORMAT * *

This message usually indicates that there is something physically wrong with the diskette itself. If you are not preserving data, 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. First format a blank diskette using the "INIT a diskette" option, then use the "COPY a disk" option to backup the suspected damaged diskette.

ERROR XX READING BLOCK XX

When using the COPY option, this message appears for every block of the disk being copied that cannot be read properly. Note the block number down for future reference. If this message occurs for every block, something is wrong. Check to make sure the disk is properly inserted in the source drive.

ERROR XX WRITING BLOCK XX

When using the COPY option, this message appears for every block of the destination disk that cannot be written properly. This message should not occur in normal operation, and indicates that the destination disk is not formatted properly. **The COPY option requires the destination disk to be previously formatted.**

HOW SECTOR SKEWING CAN AFFECT DISK PERFORMANCE

Sector skewing, or interleaving, is a term which refers to the way in which sectors are physically arranged on the track of a disk. This subject is relevant for all Apple II removable and fixed disks, but discussion here is limited to sector skewing on 5 1/4" floppy diskettes. Sector skewing is an important topic because it has a direct impact on the time required to perform a disk access (read or write).

When a 5 1/4" diskette is formatted, sector numbers are written in the Sector Address Field of each sector on each track (see **Beneath Apple DOS**, Chapter 3). For reasons which will become obvious later, we refer to this number as the **physical sector number.** You might think that sectors are numbered sequentially on the diskette, and indeed this is the normal case. All standard format programs, with the exceptions of old DOS 3.1 and DOS 3.2, lay out sectors starting with physical sector number $0 and ending with physical sector number $F, in natural order. This is why, if you analyze a normally formatted program with TRAX, the sector numbers will be in order from $0 to $F. But there is no reason that physical sector numbers **have** to be laid out in sequential order, and INIT allows you to change that order if you wish to. As we will see, changing the order can in some cases improve disk access times.

We have been careful to refer to the number in the Sector Address Field as the "physical sector number" for good reason. An important fact to understand is that **the physical sector number is not the number that the operating system refers to as the sector number.** We refer to the number the operating system calls the sector number as the **logical sector number.** It is the logical sector number, for example, that is used by ZAP. Read track 0, sector 1 of any disk in DOS format. Then change to CP/M format and read the same disk. The two "sector 1"s contain different data! This is because DOS logical sector $1 is physical sector $D and CP/M logical sector $1 is physical sector $3.

The various operating systems translate "logical sector numbers" to "physical sector numbers" using a sector translate table. (DOS 3.1 and 3.2 were written before the sector translate table existed, and in those days the logical and physical sector numbers were the same.) Figure 3.1 shows a diskette with physical sectors ordered sequentially, and shows how the operating system (DOS 3.3 in this example) references sectors by using the sector translate table.

The astute reader can now see that there are two ways that the sector ordering (the skewing) can be changed. The first way is to modify the sector translate table; that is, change the assigned physical sector for each logical sector. The second way is to write the physical sectors in a non-sequential order on the disk.

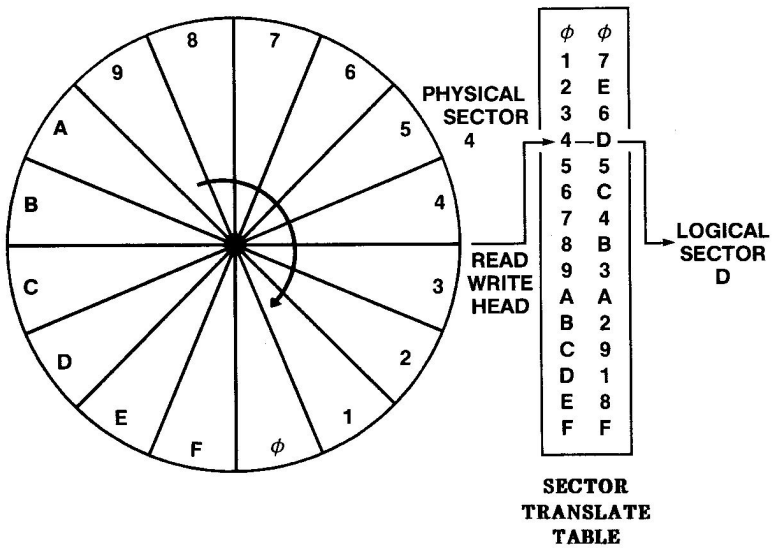


Figure 3.1 Physical Sector Order

The first method is easier to program, but it would cause chaos if you use a modified sector translate table to read a diskette that was written using a standard sector translate table, and vice-versa. The second method requires a special format program, but once the disk is formatted it will work fine with any standard operating system. This second method is the method INIT uses to change sector skewing.

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 3.2 through 3.4 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 3.1 (standard).

Shown in Figure 3.2 is the standard skewing for DOS 3.3. This arrangement of logical sectors results from a standard physical sector order and the DOS 3.3 sector translate table. The skew used for ProDOS and PASCAL is different, and still a third skewing is used for 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 the time required to boot the disk. 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 after 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).

Referring again to Figure 3.2, notice that the read/write head is positioned where it would be immediately after reading Sector F. Let's say we are doing a BLOAD, and the next sector we want to read is sector E on the same track (the usual case). After RWTS (DOS's Read Write Track Sector routine) has read all the data in sector F, control is returned to the file manager. The file manager processes the data just read, determines which sector must be read next, and then calls RWTS again. In the meantime, the diskette continues to spin in the drive. Several sectors pass beneath the read/write head before the file manager is ready to request another read. In fact, on a standard DOS

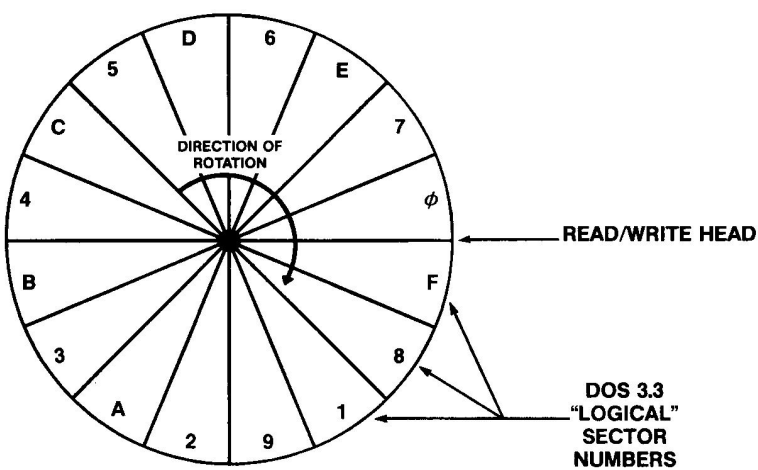


Figure 3.2 Logical Sector Order, Standard DOS 3.3

disk, the next lowest numbered sector on the track (the one we most often want to read) will be one of those sectors which has already passed by. Figure 3.3 shows the position of the diskette relative to the read/write head when the file manager is ready to read sector E.

RWTS will now look at each sector on the diskette until it 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 3.4 shows the read/write head in position to read sector E and identifies the rotational delay. This delay is wasted time.

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).

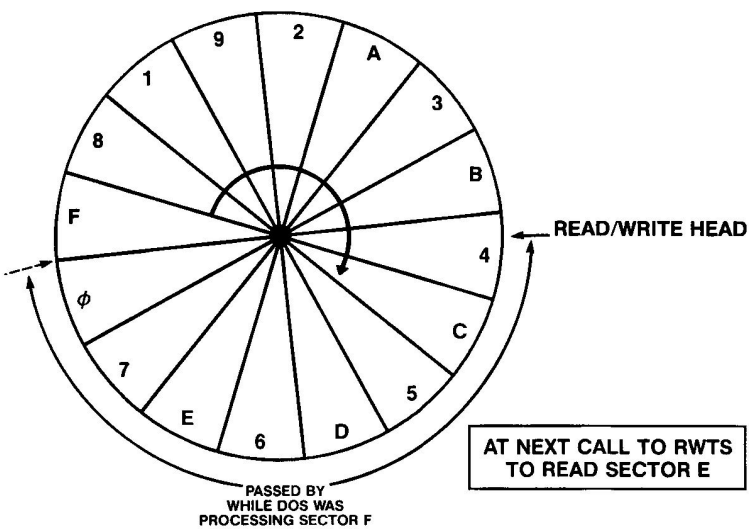


Figure 3.3 Normal Skew After Processing Sector F

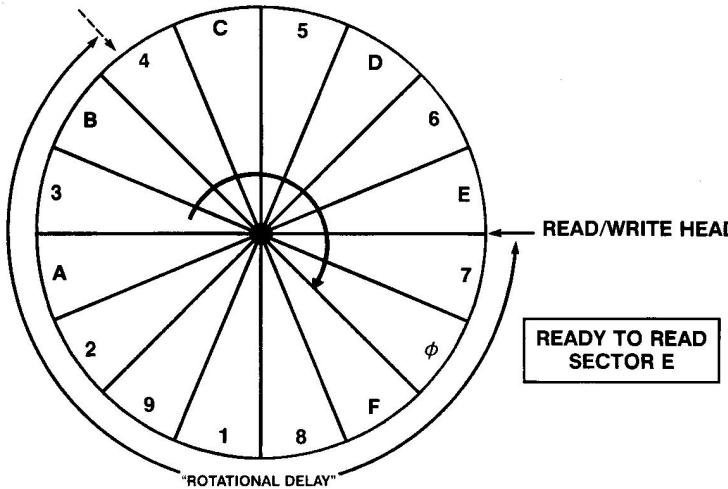


Figure 3.4 Normal Skew After Rotational Delay

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, the first method discussed above, would be to change the sector translate table to a more appropriate pattern. This works but has two major disadvantages. As we discussed before, once this table is changed any diskette created using the standard sector translate table is not accessible using the modified 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 sector translate 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 0, 1, and 2 can be optimized for booting and the remaining tracks can be optimized for LOADing. And disks created in this way may be read by any DOS 3.3 with a standard sector translate table.

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 3.5 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 3.6 shows the position of the diskette at this time.

The actual rotational delay is less than one sector long, as shown in Figure 3.7.

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

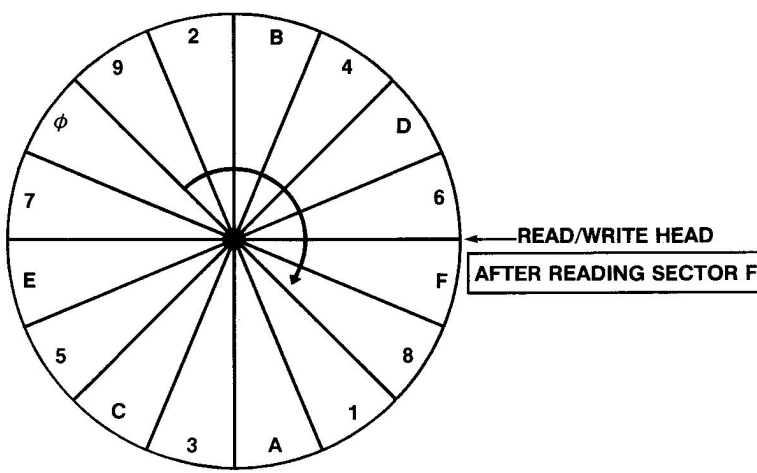


Figure 3.5 Logical Order for a "9 Descending" Skew

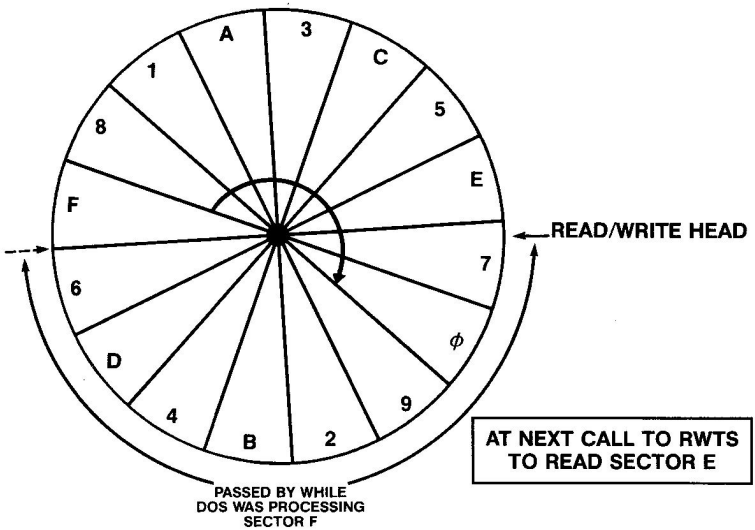


Figure 3.6 9 Descending Skew After Processing Sector F

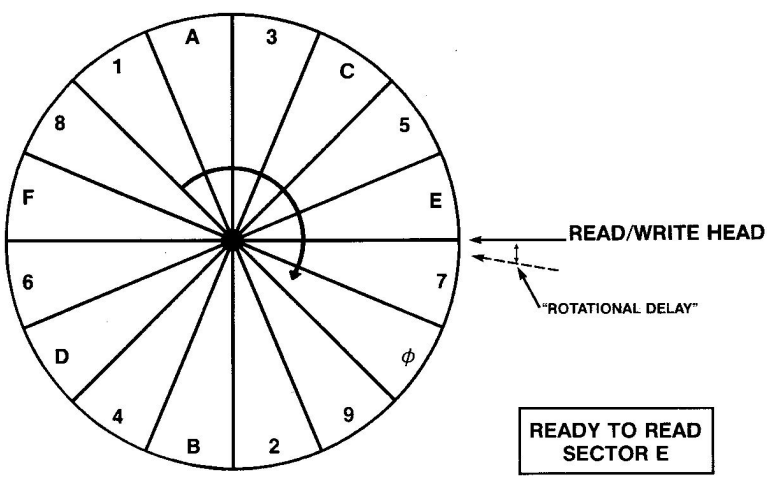


Figure 3.7 9 Descending Skew After Rotational Delay

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:

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

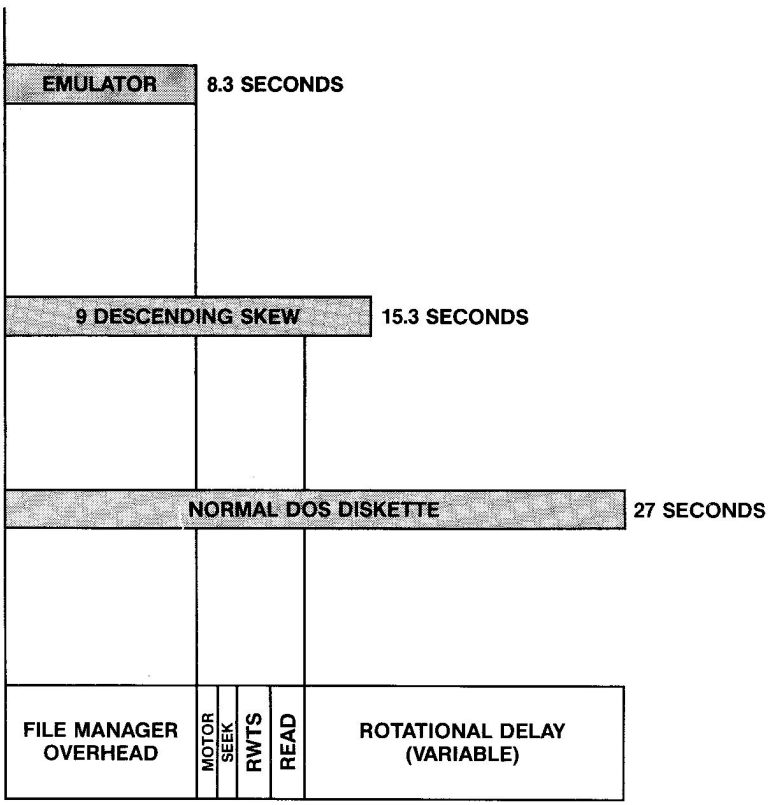
= 2 lost seconds per track

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! Obviously, the more sectors accessed the greater the rotational delay. The chart in Figure 3.8 shows the times required to BLOAD a DOS 3.3 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 3.8 shows that rotational delay amounts to more than 50 per cent of the time required to load the file with standard DOS 3.3 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 3.3 file manager overhead is quite high. This high overhead has resulted in several popular substitutes for DOS 3.3 which improve this overhead and reduce rotational delay time. The Disk II Device Driver used by ProDOS is extremely efficient compared to standard DOS 3.3.

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 3.9 shows the difference between the standard 2 DESCENDING skew and a 9 DESCENDING skew for typical DOS 3.3 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 2 INIT program, to find the best one for his particular application.



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 3.8 Comparison of Times to BLOAD a File of $7000 Bytes

	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 3.9 Timing for Typical DOS 3.3 Disk Activities

Whereas the normal skewing on a DOS 3.3 diskette is far from optimal, ProDOS, Pascal, and CP/M appear to be optimally skewed. ProDOS and PASCAL, which read 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 ProDOS, 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.

AN INIT TUTORIAL

The following tutorial will serve to introduce you to INIT's features. Boot the Bag of Tricks 2 diskette (see Chapter 1 for loading instructions), press I to select INIT, and follow along with the tutorial.

Upon entry to INIT you will be presented with a menu that gives you two choices. Select the choice, "INIT A FLOPPY DISKETTE." The screen will change and you will be presented with a display that looks like Figure 3.10

```
INIT                    INIT PARAMETER ENTRY
V2.Ø                ESC: SELECT INIT OR COPY
_____________________________________________

   SECTORS/TRACK 16_   16
     OPER SYSTEM       DOS
   PRESERVE DATA       YES
 SKEW DIRECTION        DESCENDING
    SKEW FACTOR        Ø2
          SLOT         Ø6
         DRIVE         Ø1
  VOLUME NUMBER        DEFAULT
 STARTING TRACK        ØØ-DEC   ØØ-HEX
   ENDING TRACK        34-DEC   22-HEX




_____________________________________________

Press RETURN to accept default, or
enter new value.
```

Figure 3.10 INIT Parameter 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.

Find the cursor on the video screen. The cursor is a blinking underscore which follows the first number 16. Press RETURN. By doing so you have accepted the value 16. The default values for all the INIT parameters are displayed in inverse. For each parameter you will be offered the choice of accepting the default value or typing in a new value. The cursor should now be between D and DOS, ready for your choice for the second INIT parameter.

In this tutorial we will format a blank diskette for use by DOS 3.3. Therefore, press RETURN to accept DOS as the operating system. The cursor will move down again and now offers the default "Y" meaning YES,

preserve data. But we have no data to preserve on our blank diskette, so we instead type a capital N. The letter N replaces the letter Y. Press RETURN to accept our edited input, and notice that the value in the rightmost column changes to NO. (It is possible to format a blank diskette by answering YES to this question, but it would take considerably longer than necessary because 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.)

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, press RETURN to move on 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. Leave it that way for now by pressing RETURN. The next two parameters determine the drive that will be formatted. Carefully select the slot and drive that the blank disk is in.

The volume number displays a D for default. If we left it as D, INIT would use volume 254 ($FE in hexadecimal). This is the value DOS 3.3 uses if you INIT a diskette without specifying a volume number. Instead, let's set it to volume 1 by typing 1, RETURN. Then accept 0 as the starting track, but for the ending track, type in 2 and press RETURN.

You have now selected all of the INIT parameters. Please check them. You may have a different slot and/or drive, but assuming slot 6, drive 1, the parameter values should now be 16, DOS, NO, DESCENDING, 02, 06, 01, 001, 00, and 02 (ignoring HEX values). You are asked "IS ABOVE OK ?" If not, you can either select "NO" to return to the top of the parameter list or press ESC to move to the bottom of the parameter list. Even if you have not made any mistakes you may wish to try this to see how easily errors can be corrected. When the parameters are correctly set, answer YES to the "IS ABOVE OK ?" question.

Now you will see the following message:

```
EXISTING DATA WILL BE OVERWRITTEN
INSERT DISKETTE IN DRIVE 01 SLOT 06
```

The drive and slot numbers may be different, depending on the drive you are using. 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 parameter list at which time you could make corrections.

Assuming all is well, press RETURN to begin the format operation. You will hear the disk arm recalibrate. Shortly thereafter you will see a prompt line indicating that track 0 is being initialized. Then the track number changes to 1 and 2 and the task is done. The first INIT menu will appear on the screen.

You may have figured out that the disk you have just created is a strange one. It has only three formatted tracks on it, where most 5 1/4"

diskettes have 35 tracks or more. Interesting, but not very useful. But now we will make that diskette more useful by going back and formatting the rest of it.

Select "INIT A FLOPPY DISKETTE" again. We are now going to change a few parameters. Accept the first four values but change the skew factor from 2 to 9. If you read the previous section, "How Sector Skewing Can Affect Disk Performance," you will discover that the optimal skewing for booting a DOS 3.3 diskette is 2 DESCENDING and the optimal skewing for LOADing a DOS 3.3 file is 9 DESCENDING. When we finish with this diskette, it will have boot tracks (0-2) that are optimum for booting, and the rest of the diskette (tracks 3-35) will be optimum for LOADing (assuming DOS 3.3).

After changing the skew factor, accept the slot, drive, and volume parameters and then change the starting track to track 3. Now we are going to do something a little tricky. A standard Disk II is designed to work with 35 tracks, but the fact is almost all of them can read and write at least one more track. We assume you are using a 35-track drive, and we are going to try to format 36 tracks on it. Change the ending track to 35. This will end parameter entry and the screen should look like Figure 3.11.

If the screen matches Figure 3.11 (with the possible exception of slot and drive), accept YES and then, making sure the diskette we are working with is in the proper drive, press RETURN to accomplish the formatting.

```
INIT                        VERIFY PARAMETERS
V2.Ø              ESC: INIT PARAMETER ENTRY
-------------------------------------------------

   SECTORS/TRACK        16
     OPER SYSTEM        DOS
   PRESERVE DATA        NO
  SKEW DIRECTION        DESCENDING
     SKEW FACTOR        Ø9
            SLOT        Ø6
           DRIVE        Ø1
   VOLUME NUMBER        ØØ1-DEC  Ø1-HEX
  STARTING TRACK        Ø3-DEC   Ø3-HEX
    ENDING TRACK        35-DEC   23-HEX

  IS ABOVE OK ? <YES>    No

-------------------------------------------------

Type Y or N, or use arrows to select
an option, then press RETURN.
```

Figure 3.11 INIT Screen When Ready to Format Tracks 3 to 35

If you watch carefully, you will see that the message "BUILDING CATALOG" appears right after track 17 is formatted. When the operating system is DOS and data is not being preserved, INIT puts a catalog and a VTOC on track 17 so that the disk can be used by the DOS 3.3 file manager. When the last track is formatted (track 35), the program will return to the initial menu.

We now have a formatted diskette. This diskette does not contain a DOS 3.3 boot image, as would a diskette created by the DOS 3.3 INIT command. 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. If you want DOS to know about track 35, you will have to modify the VTOC using ZAP. See pages 4-2 to 4-4 of **Beneath Apple DOS** for a description of the VTOC.

You are now familiar with the INIT function of the INIT program. The COPY function of the INIT program is self-prompting and relatively easy to use. The important thing to remember when using the COPY function is to have on hand a previously formatted diskette. If you wish to practice using the COPY function, perform the following operations:

1. Using a blank diskette that has never been formatted, format it in two steps using the INIT function. First format tracks 0 through 2, then format tracks 4 through 34. Do not format track 3! This diskette will simulate a diskette where all sectors on track 3 have been damaged. Label the diskette "DAMAGED."
2. Format all tracks (0-34) of another blank diskette in the normal way using the INIT function. Label this diskette "COPY."
3. Now select the COPY option from INIT's starting menu. You will be asked to input the source slot and drive. Make the proper selection and put the "DAMAGED" diskette in the selected drive. You will then be asked to select the destination drive. Make the proper selection and put the "COPY" diskette in the drive selected.
4. With both diskettes in the proper drive, press RETURN when prompted to do so. The "data" on the "DAMAGED" disk (it is really all zeros) will be copied to the "COPY" disk. When the program tries to copy track 3, it will generate error messages. COPY reads a block of data at a time (two sectors), so eight error messages will appear, one for each block that could not be read (blocks $18 through $1F, in this case). The copy disk will have readable sectors in track 3, but they will contain "garbage" data.

This concludes the INIT tutorial. By now you should realize that INIT is a lot more than a fancy FORMAT routine. It is a valuable aid in the process of recovering damaged diskettes, and it can be used to "speed up" DOS 3.3 diskettes.

CHAPTER 4

ZAP—By Don Worth

The ZAP utility allows you to read selected blocks or sectors from a disk storage device. The information read in can then be examined and, if desired, modified and written back to diskette. ZAP provides over 50 commands and is extremely powerful. However, a small subset of its commands can be used by the novice to perform almost any operation desired.

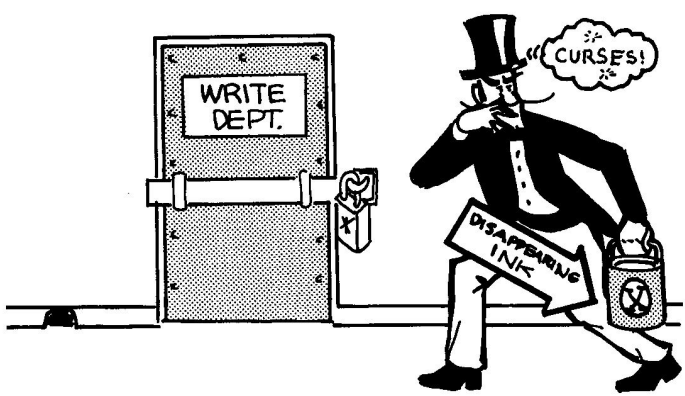
ZAP operates in one of five modes, selected by the user.

1. The DOS13 mode allows you to examine 13-sector floppy diskettes. This mode only works for 5 1/4" floppy drives (Disk II or equivalent). DOS files may be opened.
2. The DOS16 mode allows you to examine 16-sector floppy diskettes on 5 1/4" floppy drives (Disk II or equivalent). It will also allow you to examine the DOS partition of a **Sider** hard disk. DOS files may be opened.
3. The CPM mode allows you to examine 16-sector floppy diskettes on 5 1/4" floppy drives (Disk II or equivalent). It will also allow you to examine the CP/M partition of a **Sider** hard disk. CP/M files may be opened.
4. The PASCAL mode allows you to examine 16-sector 5 1/4" floppy diskettes (Disk II or equivalent), 3 1/2" floppy diskettes (Unidisk 3.5 or equivalent), and all hard disks that use block access. This includes the **ProFile** and **Sider** hard disks. Pascal files may be opened.
5. The PRODOS mode allows you to examine 16-sector 5 1/4" floppy diskettes (Disk II or equivalent), 3 1/2" floppy diskettes (Unidisk 3.5 or equivalent), and all hard disks that use block access. This includes the **ProFile** and **Sider** hard disks. ProDOS files may be opened.

ZAP defaults to the ProDOS mode. ZAP cannot be used to examine most copy-protected disks or disks that use non-standard formatting.

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 won't write to or otherwise damage your diskette. To leave the write protected mode you must issue the UNLOCK command. 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.

If you have never used ZAP before, you will want to skip over the reference materials at the beginning of this chapter and read the sections titled "ZAP--A Functional Description" and "A ZAP Tutorial." ZAP documentation is organized as follows:



ZAP DEFAULTS TO THE LOCKED MODE
TO PREVENT ACCIDENTAL WRITING

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.

ZAP COMMAND DESCRIPTIONS

DISK I/O COMMANDS

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 Bag of Tricks 2. 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. If you specify a slot that does not have a disk drive connected, the error message "NOT A DISK" will result. [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.

NOTE: Information about the capacity of the disk drives is stored by ZAP. This information is slot-dependent. When ZAP is booted, it assumes 35 tracks and 280 blocks for each device. When a command is invoked that reads the catalog or directory, such as CAT or PRODOS, ZAP will change a default value to the value found in the directory (or the VTOC, in the case of DOS). ZAP will not change the current value if it is not the default value or if the directory or VTOC does not exist. Note that the value on the disk may not be accurate, as in the case of a 35-track diskette operating in a 40-track drive. The user can always manually set the drive capacity using the TRACKS or BLOCKS command. The STATUS command will display the current value for tracks or blocks for the current slot.

TRACKS[EXP]

Set the number of tracks per drive for the current slot to [EXP]. The maximum value for tracks per drive is 50. Setting this value will also affect the number of blocks per volume for the current slot (eight blocks per track is assumed).

BLOCKS[EXP]

Set the number of blocks per volume for the current slot to [EXP]. The maximum value for blocks per volume is 65,535. Setting this value will also affect the number of tracks per drive for the current slot (eight blocks per track is assumed).

+[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 moves the cursor outside the current buffer, ZAP will read the appropriate new sector or block into the buffer. For example, if the mode is DOS16 and command +256. is issued, ZAP will read the next sector on the disk (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 operating in a track/sector mode and 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 disk or file (if in the WRAP mode).

R[EXP1],[EXP2] ..or.. R[EXP1] ..or.. R,[EXP2] ..or.. R

NO FILE OPEN, TRACK/SECTOR MODE: 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 disk or file (if in WRAP mode).
NO FILE OPEN, BLOCK MODE: Read the block indicated by [EXP1]. If [EXP2] is given, it is ignored. If no operand is given, the current block is reread. [EXP1] may be either absolute or relative. If relative, wraparound will occur at the beginning/end of the disk 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, TRACK/SECTOR MODE: Write the contents of the current sector buffer out to the disk 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.

NO FILE OPEN, BLOCK MODE: Write the current block buffer out to the disk block indicated by [EXP1]. If [EXP2] is given, it is ignored. If no operand is given, as is usually done, the buffer is written to the current block. As with the R command, [EXP1] 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 or block containing this byte.

N[EXP] ..or.. N

NO FILE OPEN: Move to the next block or sector. If [EXP] is used, add the value of [EXP] to the current block number or sector number, then read it. If no [EXP] is given, a value of +1 is assumed. [EXP] may be any absolute value from -8388608. to +8388607.. 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 block or sector. If [EXP] is used, subtract the value of [EXP] from the current block number or sector number, then read it. 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 2-byte block number or a 2-byte track/sector pair (depending on operating system mode), then reads the indicated block or sector into the buffer. This command is handy when following a chain of pointers, such as in a catalog, a directory, or a track/sector list.

%%

ProDOS indirect read command. Used only in absolute mode (no open file). The %% command looks in the sector buffer at the current offset for the low byte of a 2-byte block number and looks at the current offset plus 256 bytes for the high byte of a 2-byte block number, then reads the indicated block into the buffer. This command is designed for reading blocks from a ProDOS index block.

BUFFER COMMANDS

[EXP]

Move the buffer cursor to the offset given by [EXP]. [EXP] may range in value from zero to the end of the buffer ($FF (255.) or $1FF (511.)).

#[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).

SWAP

ZAP buffers are 512 bytes long. The SWAP command modifies the buffer by swapping the first 256 bytes with the last 256 bytes. In track-sector modes (DOS, CP/M), only 256 bytes are displayed at a time, and the SWAP command has the effect of making the "invisible" part of the buffer visible. The SWAP command is most useful when transferring data between blocks and sectors.

:[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,&,0,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.

SEARCH AND COMPARE 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 positioned to the first byte of the matched string. If an occurrence cannot be found in the buffer, the next block or 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 is partly in one block or sector and partly in another. 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 end point of the search and type a V command with the search string to be used. Position to the start point of the search 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 previously) you can read two blocks or 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

COL

Changes the display mode. If in 80-column mode, COL switches to a 40-column display. If in 40-column mode, COL switches to an 80-column display. If your computer does not have 80-column capability, this command has no affect.

NOTE: If you have an Apple II Plus with an 80-column card, ZAP will recognize it if the ProDOS loader sets the bit indicating 80-column capability in the MACHID byte (see **Beneath Apple ProDOS**, Chapter 8). The program will still need modification to work with most non-Apple 80-column cards. See the Advanced Tutorial in Chapter 6, "Modifying ZAP for Non-Apple 80-Column Cards".

LCASE

Sets the ASCII translation on the right side of the hex/ASCII display so that lower case characters will be printed "as is." The default setting for this switch is with LCASE on. LCASE only has meaning when the ASCII switch is in effect.

UCASE

The opposite of LCASE. UCASE instructs ZAP to translate lower case characters to upper case for display on the right half of the screen. UCASE may be useful to owners of Apple II and Apple II Plus computers that do not have the lower case capability. UCASE only has meaning when the ASCII switch is in effect.

IMAGE

This command only works in the 40-column mode. It sets the ASCII translation on the right side of the screen so that minimal translation is done. Inverse and flashing characters appear as is and only control characters are translated out. While IMAGE mode is in effect, UCASE and LCASE modes are ignored. The default mode is ASCII, not IMAGE.

ASCII

The opposite of IMAGE mode. UCASE or LCASE 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 not allow you to write to the disk. LOCK mode is the initial default.

UNLOCK

Resets LOCK mode allowing ZAP to write to the disk.

DOS16

Informs ZAP that the operating system is DOS and that the disk is organized into tracks of 16 sectors (32 sectors for Sider large volumes). ZAP uses this information when selecting the proper sector skewing table, setting up its RWTS (Read/Write Track/Sector) package for 16-sector I/O. When setting DOS16 mode, ZAP erases the trace table, changes the block or track/sector information for each buffer to question marks, and reads the first catalog sector into the current buffer. Subsequent use of the OPEN command will assume a standard DOS catalog is on the disk.

DOS13

Informs ZAP that a 13-sector DOS diskette is in the drive. RWTS is set for 13-sector I/O. When setting DOS13 mode, ZAP erases the trace table, changes the block or track/sector information for each buffer to question marks, and reads the first catalog sector into the current buffer.

CPM

Informs ZAP that the operating system is CP/M and that the disk is organized into tracks of 16 sectors (32 sectors for Sider CP/M volumes).

CP/M sector skewing is used. When setting CP/M mode, ZAP erases the trace table, changes the block or track/sector information for each buffer to question marks, and reads the first directory sector into the current buffer.

PASCAL

Informs ZAP that the operating system is Pascal and that the disk drive is a block device. When setting Pascal mode, ZAP erases the trace table, changes the block or track/sector information for each buffer to question marks, and reads the first block of the directory into the current buffer.

PRODOS

Informs ZAP that the operating system is ProDOS and that the disk drive is a block device. When setting ProDOS mode, ZAP erases the trace table, changes the block or track/sector information for each buffer to question marks, and reads the first block of the volume directly into the current buffer.

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 or before the beginning of a disk or a file.

NOWRAP

Turns off WRAP mode. When an attempt is made to read/write beyond the end of a disk or an open file (or before the beginning of one), 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.

SVOL[EXP]

Has no effect unless the current slot contains a Sider disk controller. If a Sider hard disk is on line, then the volume of the current operating system will be set to [EXP] as follows:

1. DOS: Selects the volume indicated by [EXP]. Small volumes are numbered first, then large volumes.
2. PASCAL: Selects the unit number indicated by [EXP]. Normally the available units are 04, 05, 0B, and 0C.
3. PRODOS: Has no effect. Use drive number to select between the two ProDOS volumes on the Sider.
4. CP/M: Selects the volume indicated by [EXP].

FILE COMMANDS

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 current operating system mode) for the file name string. The file name string may be given in hex or character (the MSB is ignored). Except for ProDOS files, 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. **ProDOS files must be fully identified,** starting with the volume directory name. A beginning slash is optional, but the name must **not** end in a slash. Subdirectories may be opened in the same manner as data files. The volume directory cannot be opened (use CAT). For example, "OPEN ASM/SOURCE/MATH" opens the file "MATH" in the subdirectory "SOURCE" on the volume "ASM" (which must correspond to the current slot and drive). "OPEN /ASM/SOURCE" opens the subdirectory "SOURCE". Files of any size can be opened if they are DOS, Pascal, or ProDOS files. If a CP/M file that is more than 256 sectors long (65,536 bytes) is opened, only the first 65,536 bytes of the file can be accessed. After opening the file, ZAP reads the first data block or 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 the size of the buffer (256 or 512 bytes). 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) or RBA (Relative Block Address) is replaced on the status line by Track/Sector or Block Number, respectively.

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 disk's directory to determine whether the current

buffer is in a file. 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 the commands listed below. 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 echoes what it is printing on the screen, this command may not work properly and the DUMP command below must be used to dump out the buffer 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 buffer in hex and ASCII onto the printer. A status line is also printed. The ASCII translation used is affected by the ASCII/IMAGE/UCASE/LCASE switches. If an expression is given, it represents the number of consecutive sectors, blocks, or records to be dumped. If no expression is given, only the current buffer is dumped. The 80-column format will be used for the dump, regardless of the current video mode.

IDUMP

Disassembles and prints the 6502 instruction equivalents for the data in the 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 ALLOW FOUR DRIVES ON ONE SLOT.

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.

MACRO COMMANDS

([NAME][TEXT])

Macro definition. The entire definition must be enclosed in parentheses and may not contain another macro's definition (no embedded parentheses). The first word is used as the name of the macro itself, and may be of any length and contain any character or number. 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. When naming macros, another consideration 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. To redisplay the hex/ASCII, press RETURN.

MSWAP

Exchanges the contents of the current buffer with those of the macro table. The buffer must contain a valid macro table, previously created by the MSWAP command, or it must be precleared to zeroes (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 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 WRITE1 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 disk (block 1), and then swap it back into the macro table. To reload the macro table, you would use the following commands:

```
R1 MSWAP
```

The MSWAP command can only be used when in a block mode (PRODOS or PASCAL) because the MACRO table is 512 bytes long.

LABEL COMMANDS

LABELS

Display all currently defined labels on the screen. To return to the hex/ASCII display, press RETURN. See the later section "ZAP--A Functional Description" for a further discussion of labels.

*

The asterisk is a pre-defined label which always contains the position following the last read or write operation.

=[NAME]

Define a new label (or redefine an old one) by setting it to 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]

Typing a label is similar to the R command. ZAP will read the block or sector associated with the label whose name is [NAME], then position the cursor at the offset associated with [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 [EXP]. ZAP then reads the resulting block or sector and positions the cursor to the resulting offset.

LSWAP

Exchanges the contents of the label table (not including the * label) with those of the current buffer. This command works almost identically to the MSWAP command, described in the section on macros. The buffer must contain a valid label table image, previously LSWAPped, or it must be set entirely to zeroes. The label table may be saved in exactly the same way as is the macro table. Because the label table is less than 256 bytes long, the LSWAP command will work in any operating system mode. 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 locations which do not exist on the current disk drive. If these labels are referenced, error messages will result.

TRACE COMMANDS

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.

MISCELLANEOUS COMMANDS

AT[EXP] ..or.. AT

The AT command is similar to the +[EXP] command listed under DISK I/O 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 block or track/sector (or relative record) information associated with the buffer, which is displayed on the command line at the top of the display. Note that if [EXP] starts with a label, it will contain block or track/sector information. See the later section "ZAP--A Functional Description" for a further discussion of the AT command.

@

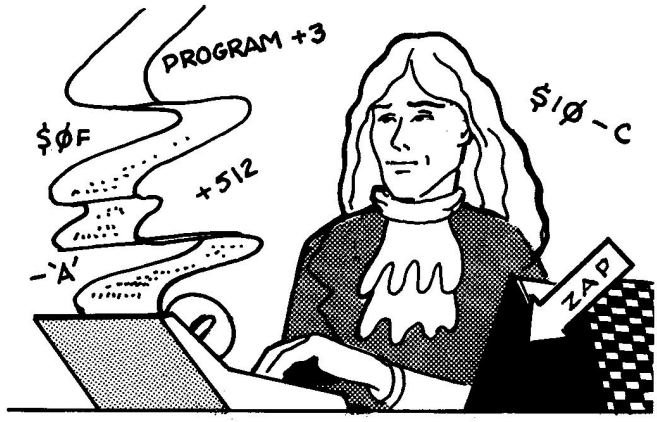
The @ character is an alias for the AT command.

?[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 lines of 6502 instructions as will fit on the screen. The disassembly begins at the current buffer cursor location. 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 the TAB key (CTRL-I) will continue the disassembly. Pressing RETURN will return to the hex/ASCII display.



THE ZAP CALCULATOR COMMAND

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). LOOP with no operands will repeat the entire command line that precedes it until an error occurs or until the user presses the keyboard. If [EXP1] is given, it is used to count the number of times the loop is to be performed. If [EXP2] is given, it is used to identify the command at which to start looping. 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. Caution is required if any macro calls are made within the LOOP. See the later section "ZAP--A Functional Description" for a further discussion of 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 6). If no expression is given, the first help screen (screen 1) is displayed. Press the TAB key (CTRL-I) to go on to the next HELP screen. Press the RETURN key to return to the hex/ASCII display.

VTOC

If the operating system mode is DOS16 or DOS13, the VTOC command will read track $11, sector $00, the DOS Volume Table of Contents. If the operating system mode is PRODOS, the VTOC command will read the block containing the Volume Bit Map (usually block 6). Issuing the VTOC command when in other operating system modes will result in a "NO VTOC" message.

CAT

The first sector of the catalog or the first block of the volume directory is read. ZAP expects the catalog or directory to be in its normal position on the disk. If what appears to be a normal catalog or directory is found, ZAP sets the volume size (in tracks or blocks) to the information found in the directory or VTOC. However, if the volume size has already been set by the user with a TRACKS or BLOCKS command, ZAP will not change the volume size.

STATUS

The STATUS command displays on the video screen several pieces of information about current ZAP status. The information presented varies somewhat depending on whether or not a file is open and on the type of operating system mode currently in effect. To exit the STATUS display, press the RETURN key.

END

The END com nand exits ZAP and returns to the Bag of Tricks 2 main menu screen.

ZAP ERROR MESSAGES

WRITE PROTECTED

Either ZAP is in LOCK mode and you have attempted to write to the disk or your disk 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 block or sector ZAP is trying to read or write. Possible problems might be:

The disk drive door is open or there is no diskette in the drive
The part of the disk you are trying to read 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 from another part of the disk. If the data in the block or 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 affected part of the diskette.

READ ERROR -- See DRIVE ERROR.

NO VTOC

The VTOC command is not valid when in the CPM or PASCAL operating system modes.

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 block or 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 disk's directory (catalog), or contains no data. Remember that ProDOS filenames must be **fully qualified** (include the volume name). This message also appears if you attempt to delete a non-existent macro or 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 block or sector which is not contained by the currently OPEN file (this is hard to do but not impossible). If you must access the block or 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 block, track/sector, RBA or RSA value. The commands, R+3 or N or P would be examples of this.

NUMBER TOO BIG

The value of an operand expression exceeds the valid range for that operand. This could happen, for example, if you tried to read track 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 normal buffer 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. ZAP only recognizes standard or well known disk devices. If ZAP does not recognize your disk drive and you think it should, please contact Quality Software.

CLOSE FILE FIRST

Some commands are not meaningful when a file is open because they would cause ZAP to read blocks or 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 or block number will be used to perform a read operation. In this case the message appears. It usually means that the data in the directory is bad (maybe you have the wrong operating system). Hopefully, you will not see this message very often.

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 disk volume or an open file 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 current 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 buffer. If you have not changed the data, then it was probably damaged when it was written to or read back from disk.

STATUS INCOMPATIBLE

In many cases, commands are valid under some situations but not under others. Some examples: the IMAGE mode is not possible when using an 80-column display, the TRACKS command is not valid when in a block mode (PRODOS, PASCAL), and the SVOL command is not valid if there is no SIDER hard disk connected to the current slot.

ZAP—A FUNCTIONAL DESCRIPTION

This section describes how to use ZAP. It is intended for first time users and for those who have not used ZAP for a while. This section also includes examples of how to use some of the more advanced features of ZAP, such as macros, labels, and loops.

If you are unfamiliar with ZAP, it is recommended that you read over this section quickly, then perform "A ZAP Tutorial" (the following section), and return to this section to study it in more detail.

Once you become familiar with ZAP, you should find the reference material in the front of the chapter sufficient to operate the program.

ZAP COMMAND SYNTAX

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:

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

will move the buffer cursor forward three bytes, store the string 'HI THERE' into the buffer at the current position, unlock ZAP to allow writes, write the modified buffer to the disk, 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!

A nice feature of ZAP allows you to re-enter the previous command with a single keystroke (RETURN is not required). If you have not pressed any keys since the last command was entered, the TAB key (CTRL-I on Apple II Plus) automatically reenters and executes the previous command.

If an error occurs during the execution of any command, processing stops and an error message is displayed. Execution of the current command and any commands following it will be aborted. Error messages are described in detail in a previous 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 pressing any key on the keyboard.

ZAP commands can be divided into several functional areas. Each of these functional areas will be discussed later in this section.

DISK I/O
BUFFER OPERATIONS
SEARCH AND COMPARE COMMANDS
OPTION SWITCHES
FILE OPERATIONS
PRINTER OPERATIONS
MACROS
LABELS
TRACE COMMANDS
MISCELLANEOUS COMMANDS

ZAP EXPRESSIONS

Wherever the string [EXP] appears in this documentation, it refers to any valid expression whose computed result will be used for the value of the operand. Do **not** type the brackets. 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** (as in Applesoft, the Apple Monitor, and DOS). If double quotes are used, the MSB is assumed to be **off** (as in ProDOS, Pascal, and CP/M). 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)

DISK I/O

Related Commands:

S[SLOT],[DRIVE]	SET SLOT/DRIVE
TRACKS[EXP]	SET TRACKS PER DRIVE
BLOCKS[EXP]	SET BLOCKS PER VOLUME
+[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[BLK]	READ BLOCK
R[REC],[BYT]	READ RECORD, BYTE (FILE OPEN)
WRITE[TRK],[SEC]	WRITE TRACK, SECTOR (FILE CLOSED)
WRITE[BLK]	WRITE BLOCK
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 OR BLOCK
%%	INDIRECT READ FROM PRODOS INDEX BLOCK

BUFFER OPERATIONS

Related Commands:

[EXP]	SET BUFFER CURSOR
#[EXP]	SELECT BUFFER
SWAP	SWAPS BUFFER HALVES
:[STR]	STORE STRING INTO BUFFER
SET[STR]	MULTIPLE STORE
&[STR]	LOGICAL AND OPERATION
O[STR]	LOGICAL OR OPERATION
X[STR]	LOGICAL EXCLUSIVE OR OPERATION

ZAP maintains sixteen (16) 256 byte areas, called buffers, into which you may read block or sector images. More than one buffer is provided to allow you to simultaneously manipulate multiple blocks or 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. Initially buffer zero is displayed.

Each buffer has associated with it a block number or a track and sector number for the image it holds. If a file is open, this number is translated into a a relative block address (RBA) or a relative sector address (RSA). This "tag" information is set whenever a DISK I/O command is issued for that buffer. If no I/O command has ever been issued or if the operating system mode (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. However, the data in an "empty" buffer remains intact to allow you to copy or compare data across different files or diskette formats.

SEARCH AND COMPARE COMMANDS

Related Commands:

L[STR]	LOOK FOR STRING
V[STR]	VERIFY STRING MATCHES BUFFER
COMPARE[BUFFER]	COMPARE BUFFERS

OPTION SWITCHES

Related Commands

COL	TOGGLE 40/80-COLUMN DISPLAY
LCASE	DISPLAY LOWER CASE AS IS
UCASE	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 OPERATING SYSTEM
DOS13	USE DOS 3.2 OR 3.1 (13 SECTOR)
CPM	USE CPM OPERATING SYSTEM
PASCAL	USE PASCAL OPERATING SYSTEM
PRODOS	USE PRODOS OPERATING SYSTEM
WRAP	ALLOW DISK OR FILE WRAPAROUND
NOWRAP	PREVENT DISK OR FILE WRAPAROUND
SVOL[VOL]	SET SIDER VOLUME

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.

FILE OPERATIONS

Related Commands:

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

ZAP normally accesses blocks by block number and accesses sectors by track/sector number. When the OPEN file command is issued, however, ZAP switches to a file relative mode. While in file mode, all references to blocks or sectors are file relative. This means that the N (next) command, for example, will read the next **logical** block or sector of the file, even if that block or sector is not the next one in order on the disk. While in file mode, blocks or sectors outside the opened file may not be referenced.

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 block or track/sector. The user can set the record length by means of the RLEN command. The default values for RLEN are 256. for DOS and CP/M files, and 512. for Pascal and ProDOS files.

PRINTER OPERATIONS

Related 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

MACROS

Related Commands:

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

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 512-byte macro table. Using the macro definition command you may store several macros in this table. At most, 511 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 block or sector in buffer zero, MREAD will read it and the next 15 sequential blocks or sectors into buffers 0 through 16, allowing you to modify multiple sectors or blocks at a time.

MWRITE Writes the contents of all of the buffers back to the disk (opposite of MREAD).

SAVEM Saves the contents of the macro table by writing it to block 1 (an unused block on ProDOS disks) on the current disk.

LOADM Reloads the macro table from block 1 of the current disk.

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

LABELS

Related 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

ZAP allows the definition of up to ten label variables. A label variable is something like a variable in BASIC. By means of the = command, the user assigns a name that is one to eight characters in length.

ZAP labels have two pieces of information associated with them:
1. a block number or track/sector number, and
2. a buffer offset.

When a file is open, the block number or track/sector number must be within the open file, and is represented by an RBA or RSA.

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 block or sector to be read and read it, leaving the buffer cursor at the appropriate offset.

Label variables can also be used in a relative way, as terms in an expression. When labels are employed in this way, the block or track/sector part of the label is ignored. Only the buffer offset value is used. The following example shows how you can use labels in a relative way.

```
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 is the first term of an expression, and is not preceded by an operator (plus or minus sign), then both the block or track/sector information and the offset information are used. If a label is not the first term in an expression, or if it is preceded by an operator, then only the offset information 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).

TRACE COMMANDS

Related Commands:

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

ZAP keeps a 32 entry trace table that records your movements within a buffer or over the disk 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 block or 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.

MISCELLANEOUS COMMANDS

Related Commands:

AT[EXP]	POSITION BUT DO NOT READ
AT	MARK BUFFER EMPTY
?[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

The AT command:

This 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 disk location which is +$3E3 bytes from the label "PROGRAM" and then write the contents of the buffer to the disk.

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 block or sector in, even if the block or 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 preceed 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 data from the disk when it sees the PROGRAM expression.

The LOOP Command:

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 * N LOOP15.
```

will fill all 16 buffers with the next 16 blocks or sectors (assuming buffer zero has just 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.

A ZAP TUTORIAL

This tutorial is intended to familiarize the novice ZAP user to some of the more basic ZAP features. Boot up a **backup** copy of the Bag of Tricks 2 diskette (see Chapter 1 for loading instructions), press Z to select ZAP, and follow along with the tutorial.

When the ZAP program is entered, it determines whether or not your computer has 80-column capability. If it does, ZAP selects the 80-column mode for you. Otherwise, of course, ZAP will begin in the 40-column mode. Those who have 80-column capability and a high-resolution monitor will probably always use the 80-column mode. Those with 80-column capability but a standard color display may prefer the clarity of the 40-column mode.

Let's begin by looking at the second line of the display. This is the command input line. The first character on this line is the prompt character (a colon). To the right of the colon is a white block, indicating the cursor. Now enter your first command by typing COL and pressing the RETURN key.

If the display was previously 80-column, it just changed to 40-column. If the display was previously 40-column, nothing happened, because you do not have 80-column capability. In either case, the word COL remained on the command line and the C is in inverse video, indicating the cursor position.

This tutorial will continue in the 40-column mode so that it will be the same for all users.

The initial 40-column screen should look like Figure 4.1. Most of the time you are in ZAP you will see screens similar to this one.

The top line of Figure 4.1 is called the status line. All values on the status line are hexadecimal values. 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.

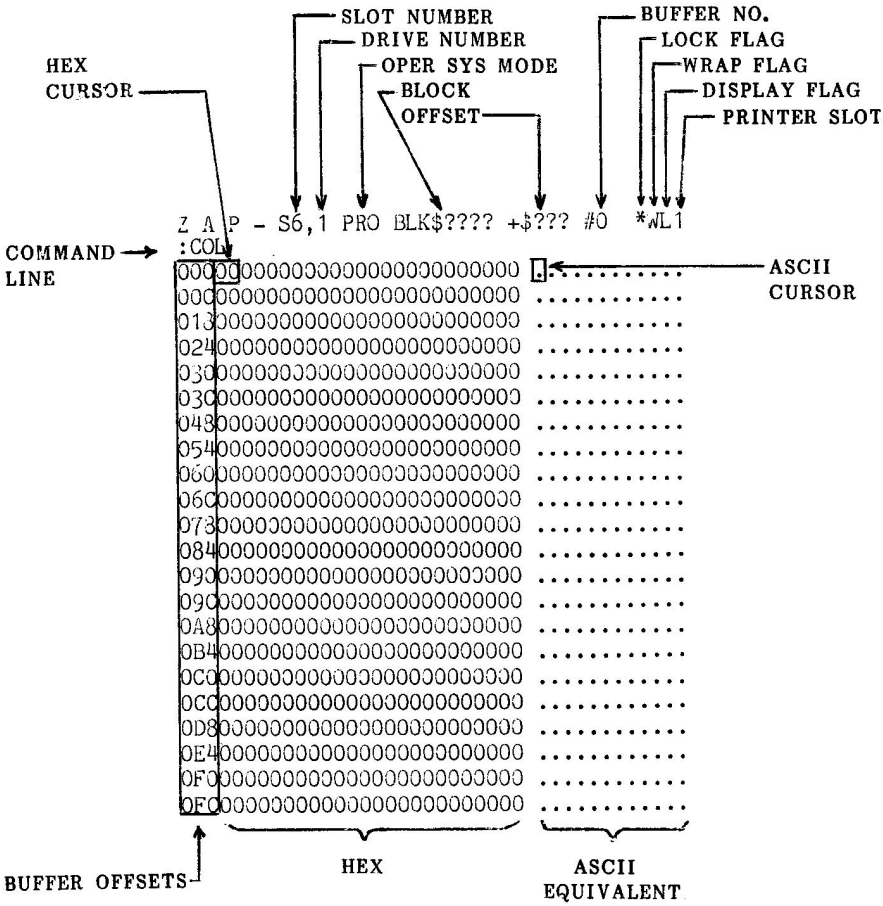


Figure 4.1 Beginning ZAP Display

DRIVE NO. Similarly, this is the last disk **drive** number used.

OPER SYS MODE The current operating system mode. ZAP defaults to the ProDOS mode.

BLOCK The last block accessed is displayed here. No block has yet been accessed, so question marks appear.

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 $0 through $F. At present, buffer $0 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. This prevents you from accidentally writing to the disk. When you are in LOCK mode, an asterisk (*) appears in this position on the screen. Otherwise, a blank appears.

WRAP FLAG This character is either W (Wrap) or N (Nowrap). The W in this case means that "wraps" are allowed. For example, if you are looking at the last block of a file and ask to see the next block, the first block of the file will be displayed.

DISPLAY FLAG 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 buffer 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.

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, in inverse video, are the hex offset locations for the first byte on each line. This is done so that you can determine the offset of any byte by counting to the right from the first on its line. Each line shows 12 bytes of hex followed by its equivalent in printable ASCII. If this were the display of the directory, for example, you would see file names over on the right hand side of the screen.

Notice that the first byte of the buffer, both its hexadecimal value and its ASCII equivalent, 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.

The buffer that ZAP displays is really just a part of the computer's memory that disk data is copied into. When data is copied from the disk to the buffer, 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 disk data in any way. Should you wish to change the disk data, you must instruct ZAP to write the contents of the sector buffer back to the disk (you don't have to write it to its original location, but this is the usual case). Thus disk data can be modified on a block by block or sector by sector basis, depending on the operating system mode.

Now it is time to actually read a block from disk into the buffer. Make sure the Bag of Tricks 2 diskette is still in the same drive and type the following ZAP command, followed by a carriage return.

```
R2
```

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 block number 2 from the disk in the current slot and drive. If you have read **Beneath Apple ProDOS** you know that block 2 of a ProDOS disk contains the Volume Directory. Your screen should look something like Figure 4.2.

Suppose you wanted to use ZAP to change the name of one of the files in the Volume Directory. This may not sound too useful, since the ProDOS RENAME command will do this. Under some circumstances illegal characters may be imbedded in the filename, 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 directory. Pretend that the file named "ZDOS" 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:

```
+A1
```

```
Z A P - S6,1 PRO BLK$0002 +$000 #0  *WL1
:R2
000000000300F4424F5432000000 ....tBOT2...
0000000000000000000000000000 ............
018000000000026AB000000000C327 ....&+....C'
0240D0900060018012650524F44 .......&PROD
0304F5300000000000000000FF OS.........
03C08001E00003A0026AB000000 .....:.&+...
043002100203 2A900000200235A .!. 2)....#Z
0544415000000000000000000000 AP..........
060000000626002B0054530059AB ...&.+.TS.Y+
06C00000000E3000858AB000002 ....c..X+...
078002742 4F5432504958000000 .'BOT2PIX...
0840000000000062900110000 20 ......)....
090000000000000000E3002025AB .......c. %+
09C000002002 45A444F53000000 ....$ZDOS...
0A8000000000000000063A0006 .........:..
0B400E3090026AB00000000E300 .c..&+....c.
0C0B5E4AA000002002454524158 5d*....$TRAX
0CC000000000000000000000006 ............
0D86C001500E4260049AB000000 l...d&.I+...
0E400E3000849AB000002002A42 .c..I+....*B
0F04F542E53595354454D000000 OT.SYSTEM...
0FC0000FF690004000005005BAB ..i.....[+
```

Figure 4.2 The Volume Directory

This tells ZAP to position the buffer cursor forward $A1 (161 decimal) bytes from its current position ($00). The cursor should now be over a $5A, the "Z" in ZDOS. Notice that the whole hex/ASCII display has moved so that the offset "024" is now at the top line of the hex/ASCII display. This is because a block contains 512 bytes, and only 264 bytes can be displayed at one time in the 40-column mode (336 bytes in the 80-column mode). When the cursor offset is in the middle part of the file, the line that the cursor is on is placed in the middle of the display. Now type:

```
:"TURKEY"
```

(be sure to type the colon). You have now told ZAP to store the string "TURKEY" 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 TURKEY. Wrong! Remember that all you have done so far is to change the data 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. In this case, we don't want to really modify the Bag of Tricks 2 diskette.

If we did, we would use the commands UNLOCK and WRITE, which would write the modified buffer out to block 2 of the diskette. Instead, type in the command:

R6

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. We are about to re-enter the Read command, but we're going to do it in a special way.

A special feature of ZAP allows you to re-enter the previous command with a single keystroke (RETURN is not required). If you have not pressed any keys since the last command, the TAB key (CTRL-I on Apple II Plus) automatically reenters the previous command line. With the R6 still on the command line, press the TAB key.

You are now looking at block number 6, which is usually the Volume Bit Map. For a description of the Volume Bit Map, see **Beneath Apple ProDOS**, page 4-5.

Now we will try some other features of ZAP. It turns out you don't need to remember the location of the Volume Directory or the Volume Bit Map. ZAP knows where they are and will find them for you if you enter either the CAT or VTOC command, respectively. Try it! When you enter the CAT command, notice that the file ZDOS is still called ZDOS. The word TURKEY never made it to the disk (thank goodness)!

Up to now you have made absolute references to blocks on the disk. ZAP will also let you "open" a file. By file we mean one of the files in a ProDOS directory. While the file is open, all references are made as "file relative". This means that, instead of asking ZAP to read a specific block on the disk, you will ask it to read a block relative to the beginning of the file. You can try this out on a very short file by opening one of the files on your practice diskette. Type:

OPEN"/BOT2/ZDOS"

This opens the file "ZDOS" on the volume directory "BOT2" (the Bag of Tricks 2 diskette). If you make a typing error, you will get a "FILE NOT FOUND" error message". Otherwise, the screen should look like Figure 4.3.

Notice that the status line has changed somewhat. The BLK information has been replaced with a relative block address (RBA). You are looking at the $0000th relative block of the file. The next block could be at the other end of the diskette, but it is always accessed by entering R1. You will not be able to read blocks outside the file while the file is open.

```
Z A P - S6,1 PRO RBA=$0000+$000 #0  *WL1
:OPEN"/BOT2/ZDOS"
0003BBD8DC0BD8EC0307C86278E @=.@=.@0|.'.
0007806AD00BB8526A9FF9D3FC0 x.-.;.&)..@
0181D8CC04868EAA00A0526208A ..@Hhj ..& .
024B588D0F3A9D52089B5A9AA20 5.Px)U .5)*
03089B5A9AD2089B598A09AD003 .5)- .5. .P.
03CB900BB59FFBAAABD66B9A627 9.;Y:*=f9&'
0489D8DC0BD8CC088D0EBA526EA ..@=.@.Pk%&j
0545900BAAABD66B9AE78069D3D Y.:*=f9.x...
060C0BD8CC0B900BAC8D0EAAABD @=.@9.:HPj*=
066C66B9A627208CB5A9DE2089B5 f9&' .5)^ .5
078A9AA2089B5A9EB2089B5BD8E )* .5)k .5=.
084C0BD8CC0601848689D8DC01D @=.@`.Hh..@.
090BCC060A9308526A04088F06D .@`)0.& @.pm
09CBD8CC010FBC9FFD007C626D0 =.@.{IP.F&P
0A8F34C87B649D5D0E9EABD8CC0 sL.6IUPij=.@
0B410FBC9AAD0F2A09ABD8CC010 .{I*Pr .=.@.
0C0FBC9ADD0E7A900888426BC8C {I-Pg)...&<.
0CCC010FB5900BBA4269900BBD0 @.{Y.;$&..;P
0D8EE8426BC8CC010FB5900BBA4 n.&<.@.{Y.;$
0E4269900BAC8D0EEBC8CC010FB &..:HPn<.@.{
0F0D900BBD014BD8CC010FBC9DE Y.;P.=.@.{I^
0FCD00BEAEABD8CC010FBC9AAF0 P.jj=.@.{I*p
```

Figure 4.3 The First Block of the ZDOS File

You can always determine whether a file is open by looking at the status line to see if BLK or RBA information is displayed.

You are now looking at the first block of a binary file. 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:

```
I
```

The hex/ASCII display has been replaced with a disassembly of the first 20 instructions. 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 block by using the I command and then the TAB key to continuously repeat the I command.

Now close the file and return to absolute mode by entering the command:

```
CLOSE
```

The BLK information reappears 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.

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. Experiment moving from block to block using the N and P commands.

You can also move ahead or back on the disk or within a file by typing a relative number of bytes that moves the buffer cursor out of the current buffer. For example, you can move forward 5,000 bytes in the file by typing:

```
+5000.
```

Try this. The plus sign makes the term relative. Don't forget the decimal point--that's what tells ZAP you are entering a decimal number. Numbers without decimal points are always assumed to be hexadecimal.

A ZAP expression can have more than one term, as long as the terms are connected by plus or minus signs. 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 our ZAP tutorial. Additional tutorial-like uses for ZAP are provided in Chapter 6.

CHAPTER 5

FIXCAT—By Don Worth

The FIXCAT utility is designed to recover damaged DOS catalogs and ProDOS directories. In the process it performs other functions such as recovering lost files and freeing up unused disk space. You have the option of directing FIXCAT's output to a printer, thereby producing a detailed report on the contents of a disk.

In normal usage, the catalog or directory areas of a disk are the most frequently read and written to. Therefore the catalog or directory are the areas of the disk that are most frequently damaged. And a damaged catalog or directory spells disaster, because none of the files listed in the damaged directory can be accessed, even if the files themselves are not damaged at all, which is the usual case. FIXCAT comes to the rescue, recovering the directory and thereby exhuming otherwise inaccessible files.

FIXCAT works by making changes to the DOS catalog and/or VTOC, or in the case of ProDOS, to directories, subdirectories, and/or the Volume Bitmap. No changes are made to the data and program files. Some errors may be detected in track/sector lists or index blocks, but these areas will never be modified by FIXCAT.

When FIXCAT works with a DOS disk, the disk must be a 5 1/4" floppy diskette. When FIXCAT works with a ProDOS disk, the disk may be any ProDOS volume, including hard disks and RAM disks.

When FIXCAT works with a DOS disk, all corrections 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.

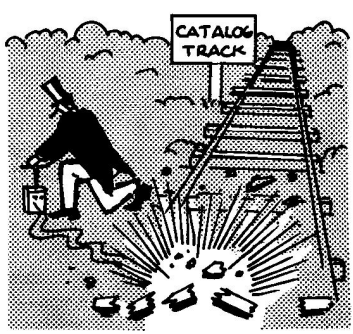
When FIXCAT works with a ProDOS disk, which may have many subdirectories in addition to the Volume Directory, the program operates on one directory or subdirectory at a time. All corrections are made to

an in-memory image of the directory being examined. Then the user is asked if he wants the corrected directory written to the disk. If he says yes, the revised directory will be written to the disk and the original directory will be overwritten. Then FIXCAT goes on to the next directory.

Note: Because FIXCAT modifies a disk (when requested to do so by the user), **it is recommended that the disk used with FIXCAT be a copy** of the original volume. The INIT program on the Bag of Tricks disk can be used to make the copy.

For **HARD DISK USERS,** it will be impossible or very inconvenient to make a copy of a damaged volume. We recommend hard disk users run FIXCAT through twice. The first time through, print out the output on a printer and always **answer NO to questions that would modify the disk.** Then carefully study the printout to help decide what questions should be answered YES when you run FIXCAT a second time. If several subdirectories are involved, you may wish to operate on only one subdirectory at a time, always generating printout and studying the printout before rerunning FIXCAT for the next subdirectory.

If you have never used FIXCAT before, you will want to skip over the section on "FIXCAT Messages" and read the sections entitled "FIXCAT--A Functional Description" and "A FIXCAT Tutorial." FIXCAT documentation is organized as follows:



A BLOWN CATALOG TRACK
MAKES A DISKETTE UNUSABLE...

FIXCAT MESSAGES

This section contains an explanation of each of the messages that FIXCAT can display. The messages are divided into those which can appear when running the DOS part of FIXCAT and those which can appear when running the ProDOS part of FIXCAT. The messages are discussed in the order that they would occur when running FIXCAT.

DOS MESSAGES

OPERATING SYSTEM SELECTED: DOS 3.x

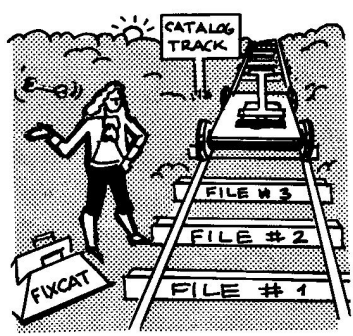
This message is sent to the printer to identify the operating system selected (DOS 3.3 or DOS 3.2).

PROCESSING DISK IN SLOT X, DRIVE X

This message is sent to the printer to identify the selected slot and drive when fixing a DOS disk.

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.



...BUT FIXCAT RESTORES THE CATALOG, AUTOMATICALLY!

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

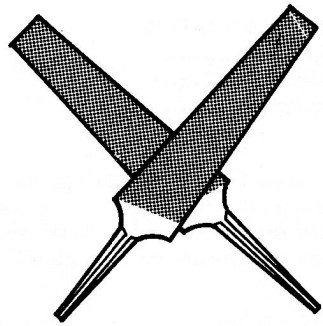
One of the track/sector pairs, describing a data sector, in the track/sector list is not valid. It is the last one printed on the DATA: message. You may have to use ZAP to correct it, assuming you can figure out what it is supposed to be.

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



OVERLAPPING FILES

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.

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:
xxxxxxxxxxxxxxxxxxxxxxx .....
xxxxxxxxxxxxxxxxxxxxxxx .....
xxxxxxxxxxxxxxxxxxxxxxx .....
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.

PLEASE SELECT THE FILE TYPE:

If you were able to identify the file, you probably know its file type as well. FIXCAT will prompt you with a menu of file types, and will supply its best guess as the default. 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.

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.

PRODOS MESSAGES

OPERATING SYSTEM SELECTED: DOS 3.x

This message is sent to the printer to identify the operating system selected (DOS 3.3 or DOS 3.2).

PROCESSING VOLUME ON UNIT NUMBER $XX

This message is sent to the printer to indentify the unit number of the ProDOS volume being processed.

I/O ERROR, PLEASE CHECK DEVICE,
THEN PRESS RETURN

Make sure the device you have selected is on and ready.

NO DEVICE CONNECTED

There is no ProDOS disk device connected to the selected slot and drive.

DEVICE WRITE PROTECTED.
PLEASE REMOVE WRITE PROTECTION,
THEN PRESS RETURN.

The device you have selected is write protected. For FIXCAT to correct errors on your disk it must not be write protected.

NOW PROCESSING /xxxxx

This is the directory FIXCAT is now processing.

BAD STORAGE TYPE, CANNOT PROCESS FILE.

The storage type for the current file is not a valid one, the file cannot be processed.

PLEASE ENTER A NEW NAME FOR THIS FILE.

The name of the displayed file is not a valid file name. Enter a new name.

ILLEGAL FILE NAME.

The name you have entered is not a valid file name. Enter a new name.

DIRECTORY NAME BAD.

The name of the directory is an illegal name. You will be prompted for a new name.

DIRECTORY NAME BAD.

The name of the directory is an illegal name. You will be prompted for a new name.

ERROR READING DIRECTORY.

When checking file names, FIXCAT was unable to read a directory block due to an I/O error.

ERROR WRITING DIRECTORY.

When writing a directory that has had a file name changed, an I/O error has occurred.

VOLUME NAME ILLEGAL.

The name of the volume is illegal.

PLEASE ENTER A NEW VOLUME NAME.

When the name of the volume is illegal you are prompted to enter a new name for the volume.

FILENAME: filename

FIXCAT is checking this file for validity and is using it to build its Volume Bitmap. The following messages may appear.

FILE TYPE: type

This is the file type found in the directory.

STORAGE TYPE: storage type

This is the storage type found in the directory.

MASTER INDEX BLOCK: $nnnn

This is the index block number found in the directory.

DATA BLOCKS: #nnnn $nnnn etc.

This is a list of data blocks for the file.

DIRECTORY BLOCKS: $nnnn $nnnn etc.

This is a list of the blocks of a subdirectory.

UNABLE TO READ ENTIRE DIRECTORY

FIXCAT is not able to process the file due to some unrecoverable error.

UNABLE TO READ FILE. DELETE IT?

FIXCAT is not able to read the first index block of a file. If you reply YES, FIXCAT will delete it from the in memory image of the directory.

TOTAL BLOCKS ALLOCATED TO FILE:$XXXX

This is the total number of blocks actually used by the file.

NUMBER OF BLOCKS USED BAD.

The number of blocks used by the file is bad. Reply YES to fix it.

VERSION NUMBER BAD.

The version number for the file or directory is bad. Reply YES to fix it.

MINIMUM VERSION NUMBER BAD.

The minimum version number for the file or directory is bad. Reply YES to fix it.

ACCESS BYTE BAD.

The access byte for the file or directory has one or more of the undefined bits set. Reply YES to fix it.

HEADER POINTER BAD.

The pointer to the directory header is bad. Reply YES to fix it.

CHECKING VALIDITY OF DIRECTORY HEADER.

FIXCAT is checking the validity of the header entry for the directory.

ENTRY LENGTH BAD.

The length of entries listed in the directory header is bad.

ENTRIES PER BLOCK BAD.

The number of entries per block listed in the directory header is bad.

FILE COUNT BAD.

The number of files listed in the directory header is bad.

STORAGE TYPE BAD.

The storage type of the directory header is bad.

BITMAP POINTER BAD.

The pointer to the Volume Bitmap in the Volume Directory header is bad.

TOTAL BLOCKS BAD.

The size of the volume in the directory header is bad.

PARENT POINTER BAD.

The pointer to the parent directory is bad.

PARENT ENTRY BAD.

The parent entry for the directory is bad.

PATENT ENTRY LENGTH BAD.

The parent entry length is bad.

UNABLE TO READ ENTIRE DIRECTORY.

FIXCAT cannot read the entire directory due to an MLI error or bad pointer.

DIRECTORY BUFFER FULL.

The directory is too long for FIXCAT to process.

ERROR READING VOLUME DIRECTORY.

There is an I/O error in one of the block of the Volume Directory.

CHECKING DIRECTORY LINKS.

FIXCAT is checking the links between the directory blocks.

LINK TO PREVIOUS BLOCK BAD.

The link to the previous block in one of the directory blocks is bad.

LINK TO NEXT BLOCK BAD.

The link to the next block in one of the directory blocks is bad.

END OF DIRECTORY LINK BAD.

The link at the end of the directory is not zero.

NO CHANGES WERE MADE TO THE DIRECTORY.

There were no changes made to the directory, so it does not have to be written to the disk.

WOULD YOU LIKE TO WRITE THE UPDATED DIRECTORY TO THE DISK?

There have been changes made to the current directory. Reply YES if you want these changes made permanent.

ERROR WRITING DIRECTORY TO DISK,
WRITE NOT COMPLETED.

There was an I/O error writing the directory to the disk, so the entire directory was not written.

ERROR WRITING VOLUME DIRECTORY.

There was an I/O error writing the Volume Directory to the disk, so the entire directory was not written.

WOULD YOU LIKE TO SCAN FOR LOST
DIRECTORIES?

If you do not think that you have any missing directories, reply NO. If you wish to have FIXCAT search the entire diskette for "unattached" directory headers, reply YES. One word of warning, however. FIXCAT may find some old directories you don't want, such as previously deleted ones. These old directories may be somewhat "moth eaten" in that some of their blocks may have been overwritten. FIXCAT will display a listing of the files in each directory that it finds, and ask you if you really want it recovered. The following messages will occur during scan.

SCANNING FOR LOST DIRECTORIES.

FIXCAT is reading every un-allocated block of the disk looking for directories. This could take quite a while for large devices, so be patient.

RECOVERY DIRECTORY FULL, SCAN ENDED.

The recovery directory buffer is full, no more files can be recovered, so the scan is ended.

WOULD YOU LIKE TO SCAN FOR LOST FILES?

If you do not think that you have any missing files, reply NO. If you wish to have FIXCAT search the entire diskette for "unattached" index blocks, reply YES. One word of warning, however. FIXCAT may find some old files you don't want, such as previously deleted ones. These old files may be somewhat "moth eaten" in that some of their blocks may have been overwritten. FIXCAT will display a portion of the first data block of each file it finds to ask you is you really want it recovered. The following messages will occur during scan.

SCANNING FOR LOST FILES.

FIXCAT is reading every un-allocated block of the disk looking for directories. This could take quite a while for large devices, so be patient.

FILE LOCATED. FIRST DATA BLOCK:

XXXXXXXXXXXXXXXXXXXXXXXX .....
XXXXXXXXXXXXXXXXXXXXXXXX .....
XXXXXXXXXXXXXXXXXXXXXXXX .....
etc.
WOULD YOU LIKE TO RECOVER THIS FILE?

FIXCAT has found an index block that is not listed in any directory. The first portion of the file is displayed so you can identify it. Reply YES if you would like to recover the file.

PLEASE GIVE A NAME TO THE FILE.

Enter a name for the recovered file.

PLEASE SELECT A FILE TYPE:

Select a file type form the list displayed.

PLEASE ENTER THE FILE TYPE: $XX

If you select other as the file type, you will get this message, aking for the file type in hex.

ERROR READING FILE.

There was an I/O error reading the file.

New subdirectory is required for
recovered files. Write it to disk?

You have recovered one or more subdirectories and/or files. It is now time to write the RECOVER subdirectory, which contains the recovered subdirectories and files, to disk. Answer YES to do this. Answer NO if you don't wish to write to disk, which will defeat the recovery process you just went through.

WRITING RECOVERY DIRECTORY TO DISK.
IF YOU RUN FIXCAT ON THIS DISK AGAIN
YOU MUST FIRST DELETE OR RENAME THE
RECOVERY DIRECTORY.

The recovery directory is being written to disk. The warning is because recovering files twice on the same disk will create two subdirectories called RECOVER unless the first one is changed before the second FIXCAT operation.

VOLUME FULL, UNABLE TO WRITE RECOVERY DIRECTORY.

There is no room on the volume to write the recovery directory. Copy a file to another disk, delete it, and re-run FIXCAT.

VOLUME DIRECTORY FULL.
UNABLE TO SAVE RECOVERY DIRECTORY.

There are no unused entries in the Volume Directory for the recovery directory. Delete a file from the Volume Directory and re-run FIXCAT to recover the files.

UNABLE TO READ OLD VOLUME BITMAP.

There was an I/O error reading the existing Volume Bitmap from the disk.

THERE WERE NO ERRORS IN THE
VOLUME BITMAP.

The existing Volume Bitmap exactly matches the one created by FIXCAT.

ONE OR MORE ERRORS HAVE BEEN FOUND
IN THE VOLUME BITMAP.
WOWULD YOU LIKE THE CORRECTED VOLUME
BITMAP WRITTEN TO THE DISK?

There are differences between the existing Volume Bitmap and the one created by FIXCAT. Reply YES to write the corrected one to the disk.

ERROR WRITING VOLUME BITMAP.

There was an I/O error writing the corrected Volume Bitmap to the disk.

PROCESSING COMPLETED.

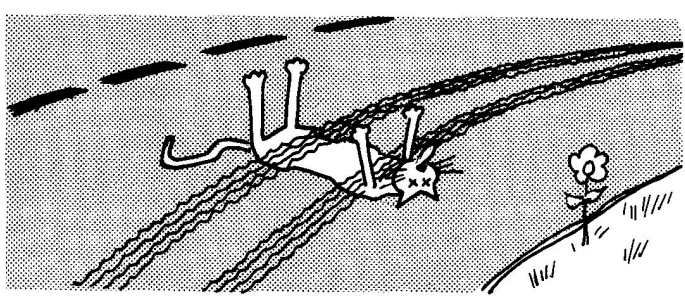
All processing of the disk has been completed, and all changes have been written back to the disk.

ERROR NUMBER $XX IN BLOCK NUMBER $XXXX

The specified ProDOS MLI error occurred trying to read or write the indicated block.

ERROR READING DIRECTORY

FIXCAT was unable to read the current directory due to an I/O error.



THIS CAT FIXCAT CANT FIX...

FIXCAT—A FUNCTIONAL DESCRIPTION

This section describes how to use FIXCAT. It is intended for first time users and those who have not used FIXCAT for a while.

FIXCAT is menu driven, and operates with a minimum amount of input from the user. When a menu selection or an input is required, the following rules apply:

Yes/No selections:

Y,y, or left arrow	Select yes
N,n, or right arrow	Select no
RETURN	Accept current selection
ESC	Go backward in program

Menu selections:

right or down arrow	Move down (or to top if at bottom)
left or up arrow	Move up (or to bottom if at top)
Number	Move to the numbered selection
RETURN	Accept current selection
ESC	Go backward in program

Input a parameter--first keystroke:
(the cursor is displayed to the right of the default value)

RETURN	Accept default
left arrow	Move cursor into default, allow editing
right arrow	Begin editing at end of default value
CTRL-D, DELETE	Edit default value, delete last character
ESC	Go backward in program
Any other key	Clear default, begin editing new value

Input a parameter--subsequent keystrokes:

RETURN	Accept entire input field
left arrow	Move cursor left within input field
right arrow	Move cursor right within input field
CTRL-D, DELETE	Delete character to the left of cursor
ESC	Cancel input, go backward in program
Any other key	Add character to input field

FIXCAT is really two programs--one that operates on DOS disks and one that operates on ProDOS disks. When you come to the third menu, "Select Operating System," you will select either ProDOS, DOS 3.3, or DOS 3.2. DOS 3.3 and 3.2 go through similar logic, operating on the catalog and VTOC of a DOS disk. ProDOS logic is quite different, however, due to the different structure of ProDOS. We have used much the same approach, however, and someone familiar with how FIXCAT works with a DOS disk will not need to learn much more to use FIXCAT on a ProDOS disk.

Printout and Timeout

The first two selections you must make when running FIXCAT are the same whether you are operating on a DOS disk or a ProDOS disk. You first must decide if you want to "LOG THIS SESSION TO A PRINTER?" The default is NO, and if you simply press RETURN, nothing will be sent to the printer and everything will be sent to the video screen.

Your inclination may be to accept the default NO, but if you do have a printer, you should consider the value of hardcopy output. Some of the detailed output that FIXCAT generates flies by so fast that it is impossible to read it on the video screen. There is always a possibility you will make a mistake while running FIXCAT (such as assigning the wrong file type) and it is helpful to have a record of what you did. Be advised, though, that if you have a lot of files on the disk FIXCAT will generate a lot of output, using up several sheets of paper. If recovering your files is worth it, we recommend YES on printer output.

If you select YES, you will be asked the following question:

WHAT SLOT IS THE PRINTER CONNECTED TO?

Turn on your printer. Enter the slot number of your printer interface (usually 1), or press RETURN to select the default value displayed. Then the program will ask,

HOW MANY TIMES IS THE WORD 'FIXCAT' DISPLAYED BELOW?

This may seem like a silly question. The computer should know how many times it printed a word on the screen. The fact is, however, that different printer interface cards operate differently, and this is FIXCAT's "foolproof" way of figuring out if your printer interface card "echos" to the video at the same time it outputs a character to the printer. Enter either a 1 or a 2, depending on what you see on your screen, and FIXCAT will know what to do to make sure you get readable output on the video as well as on the printer.

Now we are done with printer selection, and the program will ask,

AUTOMATIC TIMEOUT IN SECONDS?

If you want FIXCAT to wait for your response to each question it asks, accept the default (0). This is normally what you will want to do. If you would rather sit back and watch FIXCAT work or go away and do something else while FIXCAT is working, you can enter a non-zero value. This puts FIXCAT in the "automatic" mode. At each decision point, the program will pause for the number of seconds you set the timeout to. If you do not make a choice within the timeout period, FIXCAT will choose for you, selecting the default choice. However, if the default choice involves writing to the disk, FIXCAT ignores the timeout setting and waits until the user makes a selection.

After you have selected the timeout, the "Select Disk Format" menu will appear. You will select either ProDOS or DOS (3.2 or 3.3). We shall discuss how FIXCAT works with the two operating systems separately.

USING FIXCAT TO FIX A DOS CATALOG

The following procedures take place when FIXCAT operates on a DOS diskette:

1. The user selects the slot and drive.
2. The user inserts the diskette to be fixed and tells FIXCAT either to use the current catalog or start with a blank catalog.
3. The catalog and VTOC are checked for validity and corrections made if necessary.
4. The bit map in the VTOC is built. Space is reserved for DOS, if requested, and the track/sector lists of all files in the directory, if any, are examined and space is reserved for each existing file.
5. If the user requests, a search is made for lost or deleted files. If any are found, the first 156 bytes of the file are displayed and the user is asked if he wants to recover the file. If he answers yes, he is asked to give the file a name (or accept the name the program offers). Then he is asked to select the file type. FIXCAT offers a "guess" at the filetype, which is usually correct.
6. If the constructed VTOC bit map and the bit map on the diskette are different at this point, the user is asked if he wants to update the VTOC. FIXCAT remembers this choice, but no data is written to the disk at this time.
7. The final catalog is displayed, based on all changes requested by the user.
8. If changes to the catalog and/or VTOC are necessary, the program now asks the user if he wants to write them to the diskette. To this point the diskette has not been written to.

If an automatic timeout value was set, steps 3 to 7 are performed automatically, but steps 1, 2, and 8 will always wait for user input. You can use the ESC key to move backward through the program, and any previous changes you made to the catalog or VTOC will be reversed.

USING FIXCAT TO FIX PRODOS DIRECTORIES

The following procedures take place when FIXCAT operates on a ProDOS disk:

1. The user selects the slot and drive.
2. The user inserts the disk to be fixed and presses RETURN. FIXCAT then verifies the Volume Directory and all files in it. If the Volume Directory can be processed, FIXCAT then processes all subdirectories in the Volume Directory, and the subdirectories in those subdirectories, etc. Blocks used by all the directories and files

found in those directories are marked as in use in FIXCAT's copy of the Volume Bitmap. If the Volume Directory is not normal, the user is allowed to reconstruct it, preserving whatever "normal" data it can find there, if any. Then the user is asked if he wants to write the reconstructed Volume Directory to the disk.

3. If the user requests, a search is made for lost or deleted subdirectories. "Lost" subdirectories are common if the Volume Directory has been damaged. Any subdirectories that were in the Volume Directory are now orphans, but the information in them is very valuable. If FIXCAT finds a "lost" subdirectory, it prints the name of the subdirectory and asks the user if he wants to recover it. If so, the subdirectory is processed just as it would have been if it had been found in step 2. The subdirectory itself is placed in a new subdirectory called "RECOVER", which is later put in the Volume Directory.
4. If the user requests, a search is made for lost or deleted files. If any are found, the first 156 bytes of the file (the whole block if using a printer) are displayed and the user is asked if he wants to recover the file. If he answers yes, he is asked to give the file a name (or accept the name the program offers). Then he is asked to select the file type. Unlike the DOS part of FIXCAT, the program does not guess at the filetype, and if the user selects the default, the filetype will be set to BINary. This function will find sapling and tree files, but cannot find seedling files (512 bytes or smaller) because seedling files have no index block. If files are found and the user elects to recover them, they are put in a new subdirectory called "RECOVER", which is later put in the Volume Directory.
5. If files were recovered during either step 3 or step 4, the user is asked if he wants to write the "RECOVER" directory to the disk. If the answer is NO, then the recovered directories and files will be "lost" again, but the blocks that were allocated to them will not be freed. If the answer is YES, the RECOVER directory will be placed in the Volume Directory and the directory itself written to a currently free block. Note that this may overwrite "lost" seedling files on the disk.
6. Finally, if the Volume Bitmap constructed by FIXCAT is different than the bitmap on the disk, FIXCAT asks the user if he wants the new bitmap written to disk.

Like the DOS part of FIXCAT, you may press the ESC key to back up in the processing sequence. This should be done with more caution in the ProDOS part, however, because certain things that FIXCAT does cannot be in done. Particularly, if a subdirectory has already been written to disk, it cannot be "unwritten". Also, if subdirectories or files have been recovered, the blocks allocated to them in the bitmap are not freed up by moving back in the processing sequence. The safest way to use the ESC feature is to ESC all the way back to the "Insert Disk" screen.

If you recover subdirectories or files, all of the recovered files will be in a new subdirectory called RECOVER. It is important to immediately try to read these files using Apple system utilities or the software they were intended to run with (such as a word processor or assembler/editor).

Move the files out of the RECOVER directory and into the Volume Directory or other subdirectory of your choosing, then delete the empty RECOVER subdirectory. This is especially important if you intend to run FIXCAT on this disk a second time, because FIXCAT will contruct another subdirectory called RECOVER, and the Volume Directory will end up with two RECOVER subdirectories, only one of which can be accessed by the ProDOS file manager.

The algorithm that FIXCAT uses for finding lost ProDOS files is not perfect. A block that isn't really an index block may appear to be one. This can result in "ghost" files that really aren't files at all. Sometimes you can recognize "ghost" files because the first block is entirely zeros (an unlikely situation), or because if you try to recover them you get an "OVERLAPPING" error message.

FIXCAT makes several assumptions about the structure of a ProDOS volume. These assumptions are necessary because FIXCAT cannot assume that the information in the Volume Directory header is correct. These assumptions are:

1. The Volume Directory begins at block 2 and ends at block 5.
2. The Volume Bitmap begins at block 6 (immediately following the Volume Directory). This assumption is incorrect for Apple /RAM. However, we can't think of a good reason for running FIXCAT on /RAM.
3. When a damaged directory is repaired, entries in the directory headers are set to standard values, such as $27 for entry length, $0D for entries per block, and $0 for version number and minimum version number. Access is set to unlocked ($01 for a directory header, $E3 for all other entries).
4. When file entries are created, dates and times are set to zero (NO DATE).
5. When files types are recovered that require a value for AUX_TYPE, a default value is inserted. The values that are inserted are:

TYPE	VALUE
TXT	$0
BIN	$800
BAS	$801
VAR	$0
SYS	$2000

   The value of AUX_TYPE is set to $0 in all other cases. Note that for BIN, VAR, and random TXT files, the address will probably be wrong and the file will not operate properly until the user modifies this value. ZAP is the best program to do this with, because you can open the file and examine it (disassemble it if appropriate) to help you determine what the AUX_TYPE value should be.
6. When creating a file entry, the three-byte EOF (end of file) value is set to the number of data blocks in the file times 512. That is, the last byte of the last block in the file is the EOF. Because the file may actually be a little shorter than this, it might cause problems with text files and similar type files. Therefore the EOF value of such a file may have to be modified later using ZAP.

In addition to the assumptions just discussed, the FIXCAT program has some limitations when operating on a ProDOS disk:

1. The RECOVER subdirectory can only be four blocks long, so no more than 51 lost files or directories can be recovered.
2. FIXCAT cannot process subdirectories that are more than 16 blocks long.

Because of the complexity of the ProDOS disk structure, FIXCAT is very sensitive to I/O errors, and could behave unpredictably when encountering bad blocks. To assist with this problem, use the COPY feature of Bag of Tricks 2's INIT program. The COPY feature will allow you to copy one ProDOS disk to another, as long as the volumes on the two disks are the same size. I/O errors will be ignored, but reported on the screen. You should always use INIT COPY or some other copy program to make a copy of your disk and let FIXCAT operate on the copy, not the original damaged disk. Not only does this ensure that FIXCAT is operating on a disk with no unreadable blocks, but it ensures that you can start over if you do something you later decide you should not have done. Working with a copy of the damaged volume will not usually be possible for hard disk users, of course. There is some advise for hard disk users in the introduction to this chapter.

A FIXCAT TUTORIAL

This tutorial is intended to familiarize the novice FIXCAT user with the way in which FIXCAT operates. It is an example of fixing a ProDOS disk. Using FIXCAT on a DOS disk is similar and actually less complicated. There are two tutorials in Chapter 6 that are examples of using FIXCAT with DOS (RECOVERING LOST SECTORS IN THE VTOC FREESPACE MAP and RECLAIMING TRACKS 1 AND 2 FOR FILESPACE).

To start out this tutorial we first need to create a disk that has a damaged directory. To do this we need to use two other Bag of Tricks 2 programs, INIT and ZAP. So this tutorial is in a way an advanced tutorial such as the ones in Chapter 6 in that it involves using more than one of the Bag of Tricks 2 programs.

Boot up the Bag of Tricks 2 diskette (see Chapter 1 for loading instructions) and press I to select INIT. Then either format a blank diskette or find an old, normally formatted diskette that you don't need anymore. Label this disk "BOTX". This tutorial assumes you have two 5 1/4" floppy drives. If you don't, you will have to figure out the disk swapping for yourself.

The first thing we are going to do is to create a backup of the Bag of Tricks 2 diskette. From the first INIT menu, select the COPY option. Put the Bag of Tricks 2 diskette in drive 1 and the formatted diskette in drive 2. Select drive 1 as the source drive and drive 2 as the destination drive on the appropriate slot. When you are sure the proper disks are in the proper drives, press RETURN and the COPY operation will take place.

You now have a backup copy of Bag of Tricks 2, but we are immediately going to destroy it. ESC out of INIT and press Z to select ZAP. Then, just to be safe, remove the original Bag of Tricks 2 diskette from drive 1. Leave the copy in drive 2.

Now carefully input the following ZAP commands, in sequence. Do not type any blanks. Press RETURN after typing in each commmand.

```
S,2
CAT
SET0
UNLOCK
WRITE
```

What you have just done is to wipe out the first block of the Volume Directory by setting every byte in it to zero. There is now no way to catalog or run any of the files on the disk. It's a disaster that only FIXCAT can handle!

Now, finally, we are going to use FIXCAT. Put the good Bag of Tricks 2 diskette back in drive 1 and issue the ZAP command

```
END
```

The main menu reappears, and this time we select F for FIXCAT. When the first FIXCAT screen appears and the disk drive light goes off, remove the good Bag of Tricks 2 disk from drive 1 again, just to be safe.

The first question on the screen asks you if you want to "LOG THIS SESSION TO A PRINTER?" If you have a printer, answer YES by pressing Y or using the arrow keys, then press RETURN. You will then be asked to input the slot number that your printer interface card is in, and answer a question about how many times the word "FIXCAT" is printed on the screen. This tutorial will use up about 12 sheets of paper. If you don't have a printer, accept the default NO simply by pressing RETURN.

The next question presented is:

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

Notice that FIXCAT offers an answer to the question (0 in this case) just as it offered the answer "NO" to the question about printout. We call this suggested answer a "prompt." FIXCAT will almost always provide you with a prompt. This means that FIXCAT is guessing what your response will be to each of its questions. If you agree with FIXCAT, and want to use the prompt as your answer, just press the RETURN key. If you want to give a different answer, merely change FIXCAT's prompt to the answer you wish to select. There are two reasons for having prompts. One is to to help you to figure out the question by offering a typical answer. The second reason involves the AUTOMATIC TIMEOUT feature, which is the subject of the input you are about to make.

FIXCAT can run in two different modes. In the normal mode, each time a question is presented it will wait until you give it an 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, you could set a 1 second timeout value and then sit back and watch FIXCAT do its thing to your diskette. FIXCAT will only stop when it is about to write something to the disk. It never writes to the disk automatically, always waiting for the user's input. The prompt will always assume you want to write to the disk. Timeouts longer than one second 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 (in this case, 0 means an infinite timeout) by pressing RETURN. The next screen looks like this:

```
Select Disk Format
1. <PRODOS>
2. DOS 3.3 ( 16 Sector )
3. DOS 3.2, 3.1 ( 13 Sector )
```

We will operate on a ProDOS disk, so accept FIXCAT's prompt by simply pressing RETURN.

FIXCAT will now ask you to "Select the device to process:" and offer all the ProDOS devices (drives) currently on line. Select drive 2 of the appropriate slot, then press RETURN.

The next message is:

```
INSERT DISK (IF NECESSARY) AND
PRESS RETURN
```

This message is primarily for those who have only one drive, but it also serves to signal the start of the AUTOMATIC period if that mode was chosen. Up to now manual inputs have been required, but after you press RETURN the AUTOMATIC feature would take over if we had selected a non-zero time out. Press RETURN and processing will begin. FIXCAT will now try to read the Volume Directory. But as you recall, we blanked out the Volume Directory, so it should be no surprise that FIXCAT quickly prints the message "VOLUME NAME ILLEGAL" and asks you to type in a volume name.

We could accept FIXCAT's suggestion (BAD.VOLUME.NAME) by pressing RETURN, but we won't. Note that this is the name that would have been assigned to the volume had we been operating in the AUTOMATIC mode.

Type in the name BOTX and press RETURN. FIXCAT will then give us the message "NOW PROCESSING /BOTX" and start verifying the rest of the Volume Directory. Since the rest of the directory is all zeros, it will find a lot of things in need of "REPAIR." In fact it finds six things to repair, and asks you, in each case, if you want it to make the repair. Accept the YES answer for all six "repairs" and then you will see the question:

WOULD YOU LIKE TO WRITE THE UPDATED
DIRECTORY TO THE DISK?

Even the AUTOMATIC MODE would have stopped here and waited for your input, because a YES answer will cause a write to the diskette and the data currently in the Volume Directory block will be lost forever. This is exactly what we want, of course, so accept the YES prompt by pressing RETURN. FIXCAT writes the Volume Directory out to the disk. This Volume Directory consists of nothing but a directory header. There are no files in the directory at this time, but we know they are all still out there on the diskette. The next question we are asked is "WOULD YOU LIKE TO SCAN FOR LOST DIRECTORIES?" Answer YES. The whole volume will be searched for lost directories, but none will be found. After completing its scan, FIXCAT asks another question:

WOULD YOU LIKE TO SCAN FOR LOST FILES?

Answer yes and FIXCAT will start scanning again. But this time it will come back with the message "FILE LOCATED. FIRST DATA BLOCK:" and then start spewing out hex code and characters. The purpose of this display is in hopes that either the code or the characters will be familiar to you, the user. Sometimes this output quickly identifies the file, other times it doesn't help at all. Most of the time, if you were familiar with the files on this disk, you will figure out which file it is after a little study. This is especially true if you elect the printer option, because the entire first block (512 bytes) is printed out in that case.

The files on the Bag of Tricks 2 disk may change after this documentation is written, both in their order on the disk and in the first few bytes in the file. Hopefully the following information will be sufficient for you to properly identify each file you recover. Nine files should be found on the disk. Answer YES to recover each one of them. Type in a filename and file type according to the table below.

FILENAME	FILE TYPE	HOW TO IDENTIFY THE FILE
PRODOS	SYS	First byte is A5
ZAP	BIN	First three bytes are 4C3708
BOT2PIX	BIN	First byte is 7F. Lots of FF's in file
ZDOS	BIN	First byte is 38
BOT.SYSTEM	SYS	First byte is A9
TRAX	BIN	First three bytes are 4C4B0A
FIXCAT	BIN	First three bytes are 4C031C
INIT	BIN	First three bytes are 4C9220
MENU	BIN	See "MENU AND I/O" near front of file

If any other files are found, answer NO (very early versions of FIXCAT may find one "ghost" file).

When FIXCAT can find no more "lost" files, the following message appears:

> New subdirectory is required for
> recovered files. Write it to disk?

Answer YES. This will cause the subdirectory RECOVER, which up to now we have been constructing in memory, to be written to the diskette. The entry for this file will be written to the Volume Directory. So we now have one file in the Volume Directory, and that file is the subdirectory RECOVER. All the Bag of Tricks 2 files are entries in the subdirectory RECOVER. So even though we have now recovered all the files we lost, the "fixed" disk does not look like the original.

There is one more step that must be performed. Whenever we recovered a file, the blocks used by that file were reserved in a memory image of the Volume Bitmap. FIXCAT now checks this memory image against the Volume Bitmap on the diskette and finds they are different. Therefore it displays the message:

> ONE OR MORE ERRORS HAVE BEEN FOUND
> IN THE VOLUME BITMAP
>
> WOULD YOU LIKE THE CORRECTED VOLUME
> BITMAP WRITTEN TO THE DISK?

The two bitmaps are actually almost identical. Remember, we didn't zero out the Volume Bitmap, just the first block of the Volume Directory. There is just one block different--one more block has been allocated for the RECOVER file. Accept the YES prompt. The bitmap is written to the diskette and FIXCAT's processing is completed. The original FIXCAT screen (print option) is displayed.

It would be nice if our job was done now, but unfortunately it isn't. The Bag of Tricks 2 diskette we just recovered will not run Bag of Tricks 2 for a couple of reasons.

First of all, Bag of Tricks 2 files must be in the Volume Directory for the main menu program (BOT.SYSTEM) to operate properly. The files must be moved from the RECOVER directory to the Volume Directory (BOTX). This is a good idea anyway, because we would prefer to get files out of the RECOVER subdirectory and delete it, in case we ever want to run FIXCAT on this disk again. Moving the files is cumbersome in this case, because the files occupy 70% of the volume size, and we have to move a file, delete a file, move a file, delete a file, etc.

But even with the files back in the Volume Directory, the Bag of Tricks disk will not function properly. The reason is that when we zeroed out the Volume Directory we lost some valuable information--the AUX_TYPE

value. For BIN files, this is the load address. FIXCAT has no way of knowing what the load address was, so it just sticks in the value $800. This turns out to be the right answer for ZAP and TRAX, but disastrous for the five other BIN files. The correct values are:

FILENAME	**LOAD ADDRESS**
BOT2PIX	$2000
ZDOS	$B500
FIXCAT	$1C00
INIT	$2000
MENU	$1000

This concludes the FIXCAT tutorial. If you wish to change the AUX_TYPE values on the BOTX disk, use ZAP and refer to page 4-12 of **Beneath Apple ProDOS.**

CHAPTER 6

ADVANCED TUTORIALS

IDENTIFYING AND CORRECTING FORMATTING ERRORS

Errors in disk formatting are fatal. Soft-sectored disks have both an address field and a data field. If any portion of either of these fields is damaged, even a single bit out of place, it will be impossible to read the sector in question. If the error occurs in the **data field** portion of the sector, the data will be lost because the sector cannot be read. However, the sector may be reused because the sector can be written to. An error in the **address field** of a sector prevents writing as well as reading, and the sector must be reformatted.

If the disk in question is a 5 1/4" floppy diskette, you can use TRAX's verify command (V) to quickly locate I/O Errors. Simply boot the Bag of Tricks 2 diskette and select TRAX from the menu. After inserting the suspect diskette in the drive, issue the R command to read and analyze the first track (track 0). Noting any errors, issue the V command 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 issuing the V command.

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.

When TRAX has finished scanning the disk, it will try to read a the track that is one track past the end of the disk. Naturally it will not be able to read it, and will return the message UNABLE TO INTERPRET DATA. Don't worry--this is the way the Verify command is supposed to end.

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 requires 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. Only the damaged sectors are reformatted.

IMPROVING ACCESS TIME FOR DOS DISKETTES

Even though much of your work on the Apple II computer is not time critical, it is always desirable to speed up execution time. Disk access time is the major problem in a great many tasks. If you are using standard DOS 3.3 on a 5 1/4" floppy diskette, 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 DOS 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 capa-bilities. The only difficulty with reskewing is that the copy program on the DOS 3.3 system master assumes a conventional skewing and will not perform as fast on a diskette that has been reskewed.

Boot the Bag of Tricks 2 diskette and select the INIT program. Select "INIT A FLOPPY DISKETTE" from the first INIT screen. You will then see the INIT parameter screen, which looks like Figure 3.10 of Chapter 3. 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.

Press RETURN four times to accept the default values for the first four parameters (16, DOS, YES, and DESCENDING). The cursor should now be on the line that shows:

```
SKEW FACTOR  2_  02
```

The optimal skew is 9 and you should now make that selection by typing in 9 and pressing RETURN. Then enter the appropriate slot and drive. Then press RETURN to accept DEFAULT as the volume number. Now you should be on the line that shows:

STARTING TRACK 0_ 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_ 34-DEC 22-HEX

This is the correct value. Press RETURN to accept 34 (22 Hex), which is the last track on a standard 5 1/4" floppy diskette. Please check to see that you have made the correct entries. If you see any mistakes, simply use the ESC key to move the cursor to the appropriate line and change the parameter to its correct value.

When you are ready to proceed, answer YES to the "IS THE ABOVE OK ?" question. The following message will appear near the bottom of the screen.

DATA WILL BE PRESERVED
INSERT DISKETTE IN DRIVE 0X SLOT 0X

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. 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 beginning INIT menu will appear, indicating the completion of a successful initialization.

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 DOS or ProDOS binary type file. You can do this by using 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 2 diskette and select ZAP. First we will find the A and L values on a **ProDOS** diskette.

Enter the command CAT and the first block of the Bag of Tricks 2 Volume Directory will be displayed. Assume we want to know the A and L values for the binary program FIXCAT. First move to the FIXCAT entry in the Volume Directory by entering the ZAP command:

```
L"FIXCAT"
```

Then move to the part of the entry that contains the L value (EOF entry) by entering the command:

```
+14
```

The cursor is now over the first byte of a two-byte value that is the length of the file in bytes. This value is stored in the conventional way that binary numbers are stored (low byte first). The length of FIXCAT at the time this documentation was written was **$4A93,** indicated by a **93** at the cursor position followed by a **4A.** Your number may differ if FIXCAT was updated after this tutorial was written.

Now find the A value for the file by entering the command:

```
+A
```

Here, +A means move ten more bytes forward. This puts us on the AUX_TYPE entry for the FIXCAT file, which for binary files is the load address. The cursor should be on the value **00,** followed by a **1C.** This means the FIXCAT load address is **$1C00.**

Try finding the L and A values of a binary file on a different ProDOS disk. The procedure is the same--only the filename changes. If the file you are looking for is not in the first block of the Volume Directory, make sure when the Look command finds the filename that it has indeed found a file entry--the filename may appear elsewhere on the disk.

The A and L values for a **DOS** binary file are not in the catalog. They are stored at the beginning of the file and comprise a four-byte header. Put a DOS disk that has a binary file on it in the current drive. Then issue the command:

```
DOS16
```

This command changes the operating system type from ProDOS to DOS and displays the first sector of the DOS catalog. You can now use the OPEN command to find the binary file on the disk. Let's say the name of your file is BINARY FILE. Enter the ZAP command:

```
OPEN'BINARY FILE'
```

Notice that we used single quotes here rather than the double quotes used when we were looking for a ProDOS filename. That's because the high bit is on for DOS filenames. If you have entered the command correctly, you should now be looking at the first sector of the binary file

and the cursor is on the first byte of the file (offset 0). Let's say the first four bytes are:

D0 03 30 00

This means that the this file was saved with a DOS 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 load 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

There are many programs that you may want to patch by writing directly to the disk, but the program you are most likely to want to patch is DOS 3.3. ZAP comes in quite handy whenever you want to do this. The DOS image which is loaded into your machine when you boot a standard DOS 3.3 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 3.3, 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 Apple's memory, 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 use ZAP (in the DOS16 mode) to read track 01, sector 08, position to +$63 ($A963, 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. It is assumed you are operating ZAP in the DOS16 mode.

The recommended commands below include the LOG command. Whenever you patch DOS 3.3, 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.

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. This patch only works on a 48K slave diskette.

```
LOG NOTE PATCH TO PUT CURSOR ON CMD 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 earlier in this tutorial. 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
```

RELEASING UNUSED SPACE IN DOS 3.3 FILES

DOS 3.3 has a bad habit of not releasing disk space it once used but no longer needs (ProDOS does not seem to have this bad habit). With each DOS 3.3 file type (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 3.3 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.

Run the ZAP program and get into the DOS16 mode. 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 two bytes are:

```
3E 06
```

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. Now add one to the RSA and multiply that number by $100. In this case the result is $700. This 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. There are two ways you can free extra sectors. The first way is to write a program to read the file into memory, delete it, and rewrite it back to disk. The second way is to 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 beginning of track $11 by issuing the VTOC command. Then scan backwards through the catalog looking for your file with the command:

```
L'MY FILE'
```

ZAP will now be positioned to the filename 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 move past the last sector to the track/sector pair describing the first extra, unneeded sector:

```
+2
```

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

```
SET0
```

Now write the corrected TSL back to disk with the command 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 DOS 3.3 VTOC Bitmap."

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 only need to 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 message I/O ERROR while using one of your disks or if you just want to check a disk, you can use ZAP to scan the disk 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 2 diskette, select ZAP, and then, once ZAP is up, insert the disk to be checked. If the disk is a ProDOS disk, start by reading the first block of the Volume Directory:

```
CAT
```

In the process of carrying out this command, ZAP sets the number of blocks per volume to the number found in the Volume Directory. If the disk is a DOS disk, then instead of the CAT command issue a DOS16 command to change to DOS mode, then set the proper number of tracks with the TRACKS command. For example, if your drive is a 40-track drive, set:

```
TRACKS40.
```

At this point, whether your disk is DOS or ProDOS, issue the command:

```
R0
```

This reads the first block or sector. 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 block or sector on the disk has been read. 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 block or track/sector number of the error and retype the command line. If instead the command ends with the message 'NUMBER TOO BIG', 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, starting with R0.

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

Let's say you have an I/O error on one of your disks and you want 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 block or track/sector number of the error when you scanned the diskette. If you haven't done this, run through the previous tutorial before working on this one. Use the ZAP Read command to try to read the offending block or sector. If you still get an I/O error, try several times to read the data. You can also use some of the techniques for reading a bad sector listed in the Appendix.

If you finally manage to read the sector, the first thing to do is save the data somewhere else. WRITE it to a normally unused block (such as block 1) or sector (such as track $02, sector $0F). Now try to reWRITE the bad block or 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 reformat the disk. The INIT utility allows you to reformat individual sectors of a 5 1/4" floppy diskette, and also allows you to back up any type of damaged disk. The backup copy of the damaged disk will have usable "garbage" sectors where the damaged sectors were. See also the tutorial entitled "Identifying and Correcting Formatting Errors." If you succeed in writing the data back to its original place on the disk, 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 as described above. If you did manage to write the zeroed buffer, you can use ZAP to help determine what information on your disk has been clobbered.

First, determine if you are in a file by entering the WHERE command when the offending block or sector has just been read. If that block or sector is inside a file, the file will be opened and the STATUS command will tell you what file it is and where you are within the file.

If the WHERE command doesn't work due to an error, then the damaged block or sector may be part of the directory or catalog. If the WHERE command works but shows that the damaged block or sector is not in a

file, then you will have to determine if an important part of the disk was damaged (such as the boot routine or part of the DOS image), or perhaps the damaged sector is in a currently unused portion of the disk.

If the bad sector is in a file, you will want to try to patch the file back together. 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 following the damaged sector by skipping these records.

If the file is not a Text 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 more than 100 instructions.

COPYING PASCAL FILES TO DOS USING ZAP MACROS

This tutorial will demonstrate how to copy an Apple PASCAL text file to a DOS 3.3 text file. This combination has been selected merely as an example. A similar approach can be use to copy 13- sector DOS files to 16-sector DOS, or vice versa. In fact, files can be moved from any Apple operating system to any other, but a knowledge of the operating systems' directory structure and file types is necessary. The best way to transfer files from one operating system to another is by using Quality Software's other disk utility, **Universal File Conversion,** by Gary Charpentier. But it is possible to move files using ZAP. 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 CP/M or ProDOS sequential text file and change all references to the file name and the diskette type in the ZAP commands given.

Before we can transfer the Pascal file, we must create a dummy DOS file to transfer it to. ZAP does not contain any file management to speak of, so it cannot create a file or add sectors to an existing file. Thus it is necessary to save a dummy DOS file with at least as many sectors as the input Pascal 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 DOS 3.3 Files." We will start by OPENing the PASCAL file to find out how large it is. Boot up the Bag of Tricks 2 diskette, select ZAP, and, after it is up, replace the Bag of Tricks 2 diskette with the PASCAL diskette, APPLE3:

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

After you have entered the above command, you will see a STATUS screen with a value for the FILE SIZE in bytes (it is $000C00). This means 12 DOS sectors will be required, because each sector holds $100 bytes and $C is hexadecimal for 12. Now remove the APPLE3: diskette and boot a diskette that has DOS on it. Make sure it is a diskette that you want to write to, or replace it with another DOS diskette that you don't mind writing to. Then enter the following command to force DOS to create an output file with enough sectors:

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

Notice that we specified a file length that is $100 longer than the file size determined above. This insures we have enough space reserved on the disk. Now that the output file is created, reinsert the Bag of Tricks 2 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 DOS diskette that you wrote the dummy file to.

```
DOS16 VTOC L'GRAFCHARS' -1
```

You should now 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. Type in the following lines:

```
(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. SWAP WRITE
 ATSV2+512. =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 (the DOS 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 block containing text (that will be RBA$0002). Now you want to copy that block as the first two sectors of the DOS output file:

```
COPY
```

The deed is done! Now type N again to view the next block 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 DOS 3.3 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 3.3 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 DOS diskette that you can use as a practice diskette. Then 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 2 diskette and invoke ZAP. Insert your practice diskette in the drive and enter the following ZAP commands:

```
(OP1 #0 OPEN'FILE1')
(OP2 #8 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 eight sectors to be compared.

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

```
DOS16 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 eight 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
```

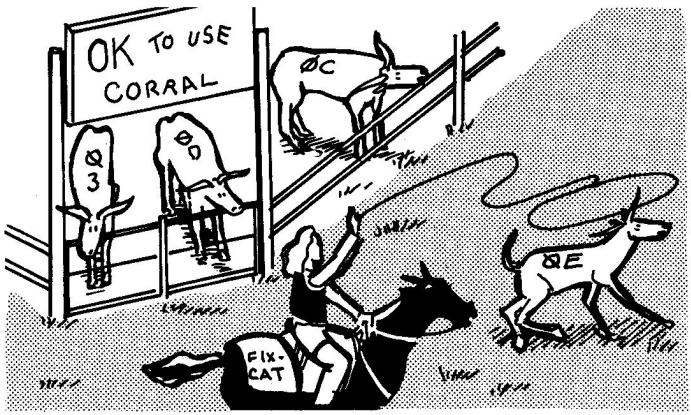
The first invocation of the CMP macro ends prematurely with a "DOES NOT MATCH" error message. The buffer cursor should be pointing at offset $3E. You do not see the text 'JUST TESTING' because you are looking at the sector which was read from FILE2. Enter #-8 to see the corresponding sector from FILE1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

RECOVERING LOST SECTORS IN THE DOS 3.3 VTOC BITMAP

The DOS 3.3 file management system does not properly "clean up" when you shorten the length of a previously saved file, and thus it is possible that a few sectors on a disk will become "lost." These lost sectors are not marked free for use in the VTOC bitmap, yet they are neither part of a file, DOS, or the CATALOG track.

When lost sectors exist, you can use FIXCAT to locate these sectors and release them for use again by updating the VTOC bitmap. 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.



RECOVERING LOST SECTORS IN
THE VTOC FREESPACE MAP

To check a diskette for lost sectors, boot the Bag of Tricks 2 diskette and select FIXCAT. Insert the diskette to be checked, select DOS 3.3 as the operating system, Read the existing catalog in, say YES to DOS image, and don't search for lost files. 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?
```

Accept the YES answer to this question, and FIXCAT will correct its 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, accept the YES answer and FIXCAT will write its corrected VTOC back to your diskette.

RECLAIMING TRACKS 1 AND 2 FOR FILESPACE

Many Apple II owners still use DOS 3.3 on 5 1/4" diskettes. It is possible for these users to add 8K of storage on most of their diskettes by getting rid of the DOS image.

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 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 cannot boot DOS from it but must boot another diskette (such as your system master) and then insert the modified diskette once DOS is loaded. 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 whenever 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 2 only supports recovery of tracks 1 and 2 as described below.

Boot the Bag of Tricks 2 diskette, select FIXCAT, and then insert a DOS 3.3 diskette that you want to use as a data disk and don't care if you ever boot again. Select the DOS 3.3 operating system, and accept all other prompts 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 NO 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. Continue through FIXCAT, accepting all prompts except answering NO to scanning for lost files. Eventually 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 bitmap does not match the old one (in which tracks 1 and 2 were marked allocated). Reply YES to this question to accept FIXCAT's changes to the VTOC bitmap. Now, when the message:

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

appears, reply YES 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 NO to the question about whether DOS exists on tracks 0, 1, and 2 on this diskette. Now perform the next tutorial to place a reminder boot program on track 0 of your diskette.

A DOS-LESS BOOT PROGRAM

Using the Bag of Tricks 2 INIT program, you can very easily produce a 5 1/4" floppy diskette that has no boot image but is suitable for data storage (operating systems other than DOS will have to have a Directory and, in the case of ProDOS, a Volume Bitmap added to them).

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 an error message. There is a simple way to avoid that problem, a way that will work effectively on any 16-sector diskette as long as sector 0 on track 0 is unused.

When a 5 1/4" floppy diskette is booted, the firmware 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 there is no DOS 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 putting a short program on your disk. Boot the Bag of Tricks 2 diskette and select ZAP from the menu. When ZAP has displayed its initial screen, insert your already formatted DOS-less diskette in your disk drive. Enter the command DOS16 and use the S command to set the slot and drive to the 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 6.1. Please verify that you have entered the data correctly before proceeding.

```
DUMP  - SLOT6,DRIVE1  DOS16   T$00 S$00 +$00  BUFF#0    LOCKED    WRAP  LC  PR1
   000 01 A6 2B BD 88 C0 20 58 FC A9 25 85 00 A9 08 85   000 .&+=.@ X¦)%..)..
   010 01 20 18 08 A9 00 F0 FC A0 00 B1 00 F0 06 20 F0   010 . ..).p¦ .1.p. p
   020 FD C8 D0 F6 60 00 00 00 00 00 00 00 00 00 00 00   020 }HPv`..........
   030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   030 ................
   040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   040 ................
   050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   050 ................
   060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   060 ................
   070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   070 ................
   080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   080 ................
   090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   090 ................
   0A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   0A0 ................
   0B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   0B0 ................
   0C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   0C0 ................
   0D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   0D0 ................
   0E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   0E0 ................
   0F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   0F0 ................
```

Figure 6.1 The DOSless Boot Program

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.

```
:'THIS IS A NON-BOOTABLE DISK' (return)
:8D (return)
:'BOOT FROM ANOTHER DISK' (return)
```

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

When you are satisfied that you have entered the information correctly, you may write the buffer to your disk. 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 DOS 3.3 FILE

There are two ways to un-DELETE a DOS 3.3 file you have mistakenly 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 and CATALOG it. Pick out the least important file and DELETE it. Then boot the Bag of Tricks 2 diskette and select ZAP. Replace the Bag of Tricks 2 diskette with the DOS diskette and enter:

```
DOS16
```

This will position you to the catalog. Now type:

```
VTOC L'filename'
```

where "filename" is the name of the file you just deleted. This will find the catalog entry for the deleted file, which still exists because no files have been added to the disk since we deleted it. 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. The $FF indicates that the file has been deleted. It must be replaced with the track number of the first Track/Sector List for the file. Conveniently, DOS always saves this number in the last byte of the file name before changing it to $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 DOS 3.3 VTOC Bitmap," 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 bitmap 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 lost or deleted files. If the Track/Sector List for the lost file hasn't been overwritten, FIXCAT can 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 2 diskette and select FIXCAT. Select DOS 3.3 as the operating system, and accept all other prompts. Insert the diskette to be searched when FIXCAT asks for it. (You can try this out on the same diskette you worked on above by DELETEing the same file again.) When the message:

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

appears, accept YES. 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:
xxxxxxxxxxxxxxxxxxxxxxx .....
xxxxxxxxxxxxxxxxxxxxxxx .....
xxxxxxxxxxxxxxxxxxxxxxx .....
xxxxxxxxxxxxxxxxxxxxxxx .....
etc.

RECOVER THIS FILE?
```

If you think this might be your long lost file, reply YES. 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 YES to finish the un-delete process. Note that you will probably see the message:

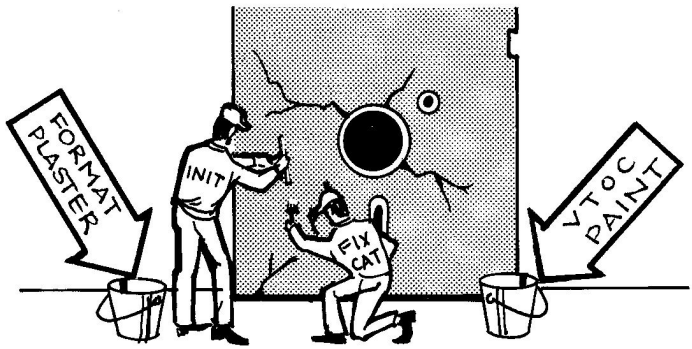
```
ONE OR MORE ERRORS WERE FOUND IN THE
VTOC BIT MAP.  CORRECT THEM?
```

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

RECONSTRUCTING A BLOWN CATALOG

It's a sad day when the catalog or directory of the disk you are working with is damaged. This catastrophe is usually indicated by an I/O ERROR message when you try to CATALOG the disk. If this happens, you will be unable to do much with the disk. And, as many users realize, it's very likely that all the files themselves are in perfect condition. But with a damaged directory, there's no way to use them.

If you are in this predicament, don't lose hope. This is the time for FIXCAT to step into the phone booth, shed its drab "disk utility" clothing, and step out a genuine computer hero. In "A FIXCAT Tutorial" in Chapter 5, we showed you how FIXCAT can be used to repair a blown Volume Directory on a ProDOS disk. In this tutorial we will show how FIXCAT can be used to recover a blown DOS catalog.



INIT & FIXCAT WORK TOGETHER TO REPAIR DISKETTES

To perform the tutorial it will first be necessary to create a damaged catalog on a practice diskette. To do this, boot the Bag of Tricks 2 diskette, select ZAP, and then insert a diskette that has a few not-too-valuable files on it. Now type:

```
DOS16 VTOC
SET0 UNLOCK WRITE N LOOP16.
```

The first line will position you to the VTOC of the practice diskette. The next set of commands changes each byte of each sector of the catalog track, including the VTOC, to zero. 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 the previously 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 make a backup copy of the damaged diskette. The copy of the damaged diskette can then be operated upon as described by this tutorial.

Now that you have a DOS disk with a totally destroyed catalog track, it is time to run FIXCAT. Boot the Bag of Tricks 2 diskette and select FIXCAT.

FIXCAT will first ask if you want printer output. Accept the NO prompt. When asked for the automatic timeout value, reply 0 (no timeout). Next select the DOS 3.3 disk format, and select the appropriate slot and drive. 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

WOULD YOU LIKE IT REPAIRED? <YES> NO
```

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. Press RETURN to accept the YES prompt on each "...REPAIRED?" question until you see the messages:

```
CHECKING FORMAT OF CATALOG...

T$11 S$0F
 CATALOG LINK BAD
WOULD YOU LIKE IT REPAIRED? <YES> NO
```

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

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

          <YES> NO
```

If you have never "wiped" DOS from this diskette (as described in a previous tutorial), reply YES (accept the prompt). Otherwise, reply NO. Now you will see this question:

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

     <YES> NO
```

Accept the YES prompt. 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
00000000000000000000000000 ............
00000000000000000000000000 ............
00000000000000000000000000 ............
00000000000000000000000000 ............
00000000000000000000000000 ............
00000000000000000000000000 ............
00000000000000000000000000 ............
00000000000000000000000000 ............
00000000000000000000000000 ............
00000000000000000000000000 ............
00000000000000000000000000 ............
00000000000000000000000000 ............

RECOVER THIS FILE? <YES>  NO
```

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 assume that no files were ever deleted, so accept the YES prompt. 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 by typing in whatever name you wish. For the present, accept the prompt of UNKNOWN FILE #00. We will change it later with DOS's RENAME command.

Now FIXCAT will display some more information about the file, such as:

```
FILE: UNKNOWN FILE #00

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

TOTAL SECTORS ALLOCATED TO FILE: $0002
```

In this example, 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 two sectors (counting the TSL itself).

Now FIXCAT asks you to select the file type:

```
Please select the file type

1. Text
2. Integer Basic
3. <APPLESOFT BASIC>
4. Binary
5. Relocatable
6. S type file
```

Notice that FIXCAT has guessed that the file is an APPLESOFT program. FIXCAT's guess at DOS file types is usually correct. If the disk you are working with in this tutorial is less than half full, the first file FIXCAT will find will probably be the HELLO program, and that program will probably be written in Applesoft. If this is indeed the fact, FIXCAT probably guessed Applesoft BASIC on your disk, too. In any case, accept FIXCAT's suggestion.

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 is finished and all the lost files have been found, you will see a message similar to this one:

```
 72 SECTORS IN USE
488 SECTORS FREE

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

            <YES>  No
```

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 YES to cause FIXCAT to substitute the computed bitmap for the zeroed one. Now a CATALOG will appear on the screen, showing all the files that you have recovered. This is helpful in reviewing the work just done. Press the SPACE BAR when you are ready to continue. Then you will see:

```
PROCESSING COMPLETED.

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

Reply YES to this question to cause FIXCAT to write back its reconstructed catalog track to the practice diskette. FIXCAT's job is now done. Boot the DOS disk or, if the disk is not bootable, boot DOS with some other disk and put the practice diskette back in the drive. It is time now to try to identify the files and rename them to something

more sensible. First LOAD in the files that have been identified as Applesoft BASIC files, one by one. LIST each one and decide what to name it. Then use the RENAME command to change the name. For example:

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

You may be able now to identify the other files, if any, by their file type or file size in sectors. If not, the ZAP program is about the easiest way to examine Text files (the ASCII display is helpful) or Binary files (ZAP allows you to disassemble files with the I command). Before long you will have the diskette looking just like it did before we zeroed out its catalog track.

MODIFYING ZAP TO WORK WITH A VIDEX 80-COLUMN CARD

ZAP's 80-column mode is designed to work on an Apple IIe with an 80-column card or on an Apple IIc. It probably won't work perfectly with any non-Apple 80-column card for the Apple II Plus. But ZAP has been designed so that the Apple II Plus owner can, with a little effort, modify the ZAP program to make it work with his hardware.

If you intend to modify ZAP, be sure to modify a copy of the Bag of Tricks 2 diskette. After the modification is made, ZAP will no longer operate properly with an Apple IIe or IIc. Who knows, you may buy an Apple IIe some day!

If you happen to have a Videx Videoterm 80-column card, you are in luck. We happen to have the necessary routines hidden at the end of the ZAP program. All you have to do is change a few bytes at the beginning of the program and ZAP will use the hidden Videx routines.

Boot a copy of the Bag of Tricks 2 diskette, the one that you want to turn into a Videx version. Type COL to enable the 40-column display so you can see clearly what you are doing. Then open the ZAP file by entering the command:

```
OPEN"/BOT2/ZAP"
```

Modify the first block of the file as follows:

```
04:2A5B4C345B4C3A5B4C405B4C  (return)
10:455B4C4A5B  (return)
UNLOCK WRITE END
```

The next time you run ZAP, the 80-column display on your Videx card should work correctly!

Now, what is this magic we have done? ZAP has vectors at the very front of the program that handle the usual 80-column functions. The first few lines of ZAP source code are:

```
        ORG   $800
ENTRY   JMP   ZAP
CLRVEC  JMP   CLEAR     Clear 80-column screen
INVVEC  JMP   SETINV    Set inverse mode for 80-column
NORMVEC JMP   SETNORM   Set normal display mode for 80-column
EOLVEC  JMP   CLREOL    Clear from cursor to end of 80-col line
EOPVEC  JMP   CLREOP    Clear from cursor to end of 80-col screen
C40VEC  JMP   COL40     Return to 40-column display
```

Normally the vectors from CLRVEC to EOPVEC are set to Apple monitor locations, but the 80-column firmware, which was developed after most Apple II Plus video cards were already designed, was designed to run only with Apple hardware. Therefore these jumps must be changed to jump to routines which will perform the desired function.

If you are a machine language programmer and your 80-column card came with complete and understandable documentation, you should be able to write your own 80-column routines in machine language and append them to the end of the ZAP program, just as we did for the Videx card.

The area of memory starting at $5B2A (where the Videx routines are located) is free to $5E00. It is in this area where routines for other 80-column cards should be located. Poke or load your routines into this area of memory after BLOADing ZAP, then resave ZAP to disk with a command such as:

```
BSAVE /BOT2/ZAP,A$800,L$5500
```

Then modify the first sector of ZAP just as we did above for the Videx routines, poking in the appropriate addresses.

A ZAP MACRO FOR OPENING PRODOS FILES

If you often use subdirectories, or you are going to OPEN a lot of files with ZAP that are in the same Volume Directory, then you may begin to curse ZAP because it always insists you enter the fully qualified filename. You wish that there was some way to enter a default prefix.

Ah, but there is! All we need to do is to write a simple ZAP MACRO. Let's say that we are going to be looking at several files in the ProDOS volume called /ASSEMBLER, in the subdirectory /SOURCE. For example, if we wanted to open the file "MULT" in that subdirectory we would have to type:

```
OPEN"/ASSEMBLER/SOURCE/MULT"
```

That's a lot to type in! So let's create the following macro command:

```
(O: OPEN"/ASSEMBLER/SOURCE/)
```

A ZAP macro command is really no more than representing a long string with a short string. The last command in the macro can take a parameter or, as in this case, the last part of a parameter. Now all we have to do to open the "MULT" file is to enter:

```
O:MULT"
```

After the O: macro is expanded, it will look just like the OPEN command above. Just be sure to include the trailing double quote (") or a syntax error will result.

USING ZAP WITH A "SIDER" HARD DISK

This tutorial is for those who own a "Sider" hard disk. The Sider can be partitioned into four separate operating systems, DOS 3.3, ProDOS, Pascal, and CP/M, and ZAP has been designed to allow Sider owners to examine and modify data in each volume of all four operating systems. ZAP assumes that all four operating systems have been installed, and it is not recommended to try to access partitions that don't exist, because any writes under such conditions could erase valuable data from the hard disk.

With the Sider turned on, boot the Bag of Trwcks 2 diskette and enter the ZAP program. In this tutorial we will assume that your Sider controller card is in slot 7. If it is in some other slot, please substitute your slot number wherever applicable. Enter the ZAP command S7 to tell ZAP you want to work with Slot 7.

We will look at the DOS partition throughout most of this tutorial. Enter the ZAP command DOS16.

ZAP figures out that there is a Sider disk drive in Slot 7, and defaults to the first volume of the requested operating system--in this case Volume 1 of DOS 3.3. Displayed on the screen is the first catalog sector of this volume. When you change operating systems, ZAP always does a CAT command. ZAP also does a CAT command when switching volumes on a Sider hard disk. This is important to remember, as we will see later in this tutorial.

The first DOS volume on a Sider is always a "small volume" in Sider terminology, which means it is identical in size to a standard floppy disk--35 tracks, each with 16 256-byte sectors. Notice that when ZAP is working with a Sider disk that it displays the volume number (on the 80-column display) just above the ASCII display on the right side of the screen. Now do a STATUS command. The status reveals that this is a small volume ($10 SECS/TRK, $23 TRKS/DISK).

You may also have "large" DOS volumes on your disk. Large DOS volumes on a sider are volumes divided into 50 tracks, each with 32 256-byte sectors. Let's say you know that DOS Volume 9 on your Sider is a large volume. Enter the command SVOL9, then a STATUS command.

First notice the track/sector "tag" on the top line of the screen. You should be looking at Track $11 (the Catalog track), Sector **$1F.** The catalog track on these large volumes is 31 sectors long rather than 15 sectors in a small volume. Also notice that the STATUS display for the large volume shows $20 SECS/TRK and $32 TRKS/DISK.

Now we will go through some tasks you can do with ZAP on your Sider disk. You may wish just to read through the rest of this tutorial without actually performing it. Bear in mind that we are not working with some "practice" disk here, we will be working with your (probably one and only) Sider hard disk. Be warned, therefore, that if you make an unlucky typing error it could result in erasing valuable data from your disk.

The first task we are going to demonstrate is to copy one small DOS volume to another small DOS volume. This is not the same as backing up your hard disk, of course, because if the Sider should totally fail you will use both volumes. But often you have enough room on the hard disk that a volume is sitting there completely empty. It might as well contain a copy of your most important DOS volume in case you accidentally erase or otherwise fatally modify one of your files.

First select a volume on your hard disk that is empty. Confirm that it is a small, empty volume by looking at the first sector of the catalog and verifying that first three bytes are 00 11 0E and that the remaining bytes in the sector are 00. We are going to assume that Volume 4 is such a volume. Thus the command SVOL4 displays a catalog sector such as we have just described.

Now select the volume that you wish to copy. We are going to assume that the volume you want to copy is Volume 1. Given these assumptions, the following ZAP macro command should be written:

```
(TRANS #1 SVOL4 #0 WRITE #1 SVOL1 #0 N)
```

This creates the macro TRANS which switches to Volume 4, copies the current sector to that volume, then returns to Volume 1 and increments to the next sector. Notice that we change to buffer #1 whenever we move from one volume to another. This is because the CAT command is performed as part of the SVOL command, and would change our "tag" information in buffer #0, thereby confusing the N command.

Now we make sure we are ready to start the transfer by accessing the first sector of Volume 1:

```
DOS16 SVOL1 UNLOCK #0 R0
```

We haven't done any transferring yet. If you want to perform this exercise, a good idea would be to issue the MACROS command to see the TRANS macro you just typed in displayed. Make sure it corresponds exactly to the definition above. Then perform the transfer by issuing the ZAP command string:

```
TRANS LOOP560.
```

This transfer takes about three and a half minutes. It is not the most efficient transfer because it reads one sector, then writes one sector, for all 560 sectors in the volume. When it is done, it should display Sector 0 of Track 0 of Volume 1. You may wish to inspect Volume 4 with ZAP to see that the transfer was indeed made.

Perhaps even more useful would be a routine to copy a DOS volume to a floppy diskette, thereby "backing up" part of the hard disk. At this writing there are no real good backup programs written for the Sider, but there will very likely be some excellent backup programs written in the future. These programs will be much easier to use that ZAP, but there is no reason why you can't use ZAP to perform backups to floppy disks, if you are careful. The following example shows how a small DOS volume from a Sider can be transferred to a standard floppy disk using ZAP, and some of these techniques may be used by skillful programmers to copy large DOS volumes and other operating system volumes to multiple floppies.

The following ZAP macros may be used to copy Sider small DOS volumes to and from a floppy disk, for backup purposes:

```
(NXTR #+1 *+256. R)
(R4SEC R NXTR NXTR NXTR)
(W4SEC #0 WRITE #1 WRITE #2 WRITE #3 WRITE)
(BETWEEN1 =SAV TOHARD #0 ATSAV+256.)
(BETWEEN2 =SAV TOFLOP #0 ATSAV+256.)
(TOHARD #4 S7 SVOL4)
(TOFLOP S6,1)
```

The first five macros are independent of the slot and drive numbers and the volume number of the Sider small DOS volume. The last two macros, TOHARD and TOFLOP, should be changed if necessary. As written, TOHARD assumes the Sider controller is in slot 7 (S7) and the Sider small DOS volume of interest is Volume 4 (SVOL4). Change the S and SVOL parameters if this is not the case. As written, TOFLOP assumes slot 6, drive 1 for the 5 1/4" floppy drive. Change it if this is not the case.

To prepare to backup the Sider volume (**download**), you must first format a DOS diskette. This diskette does not have to have any catalog or DOS information on it, but all sectors must be formatted. All sectors on the floppy will be written over by the corresponding sectors on the hard disk volume.

When a formatted floppy is ready in the floppy disk drive, enter the ZAP command string:

```
DOS16 TOHARD UNLOCK #0 R0
```

This displays the first sector of the Sider small DOS volume (usually all zeros). Check the macros you typed in and then enter the command string:

```
R4SEC TOFLOP W4SEC BETWEEN1 LOOP140.
```

This loop reads four sectors from the hard disk, then writes them to the floppy disk, then repeats this process 140 times. The process takes just under three minutes.

To restore the floppy information to a Sider small DOS volume, you must first identify a small DOS volume that is empty or has absolutely no files or other information in it that you need any longer. The **upload** process will overwrite everything in the small DOS volume identified in the TOHARD macro.

With the proper floppy disk inserted (the one with the previously saved Sider small DOS volume on it), and the TOHARD and TOFLOP macros checked to be sure they are correct for the desired transfer, type in the following command string:

```
TOFLOP DOS16 UNLOCK #0 R0
```

This displays the first sector on the floppy disk. Now enter the command string:

```
R4SEC TOHARD W4SEC BETWEEN2 LOOP140.
```

This command reads four sectors at a time from the floppy and writes them to the selected hard disk volume. The upload process takes about twice as long as the download process, for reasons we have not yet been able to determine.

ZAP Sider routines were based on the best information available from the Sider developers, First Class Peripherals, who were very cooperative in this effort. The Sider software is in a continuing state of development, however, and future Sider products may or may not be compatible with ZAP. We welcome comments from Sider owners.

APPENDIX A

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 disk I/O 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 files!** We have learned by experience that no matter how careful you are, sooner or later you will encounter an I/O error on a disk. Most errors result in the non-recoverable loss of at least some data on the disk, and a few errors may cause entire tracks or even entire disks to become non-recoverable. If you have valuable data on your disks, we recommend heeding the warning of the pessimists who maintain that your most vital data will always fail first.

SYMPTOMS AND POSSIBLE SOLUTIONS

Listed below are some of the symptoms you are likely to encounter if a diskette has been damaged in some way. In each case the likely cause of the error is given and a possible solution is suggested. The subsection 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, the disk you are trying to boot may be an old 13-sector diskette. Second, track 0 of the disk may be damaged. Most likely at fault is sector or block of track 0. If you know what operating system the disk is supposed to use, try putting a new copy of the boot routine for the appropriate operating system on the disk. For example, if it's a DOS disk, put a new copy of DOS on track 0.

Disk doesn't boot (recalibrates)

Something is wrong with the boot process. Either the boot logic is damaged or the operating system itself is damaged or the disk contains no operating system. Try installing a fresh copy of the operating system.

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

During the boot process, the operating system typically reads the catalog or directory, finds a file entry, and loads and executes that program. For example, during a DOS 3.3 boot, 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 where the problem lies, boot another disk that loads in the appropriate operating system. Then try to do a directory (or catalog) of the suspected disk. If that works, then try to load in the file or files that should be loaded during the boot process.

The message "* UNABLE TO LOAD PRODOS ***" is displayed**

Given you have 64K and your hardware is functioning properly, this message means you are trying to boot a disk that has a ProDOS boot image on it, but the "PRODOS" file cannot be located in the directory. If the disk is damaged, it is most likely the Volume Directory block that is damaged.

I/O Error on CAT or CATALOG command

Either the bitmap or some part of the directory is damaged. If your disk is a standard DOS or ProDOS disk, FIXCAT should provide a solution to this problem.

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

The file itself is damaged, either in index block, track/sector list, or the data portion of the file. Use ZAP and try to open the file. This should help you further isolate the exact location of the damage.

Recalibration occurs during successful operation

While the diskette may simply be off center, it is likely that an error occurred from which the operating system 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. Backup the diskette using the COPY feature of the INIT program, and read the subsection below on "Data Errors" for information on rewriting the faulty sector or block.

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.
The second category of data errors consists of those errors that always destroy data. There are two ways that these errors manifest themselves, and hardware failure is the suspected 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 during the write process. While writing to the disk, one or more sectors is totally destroyed. An example would be to SAVE a file to disk and then discover that the bitmap and one directory block was destroyed. That data, although not readable, may not necessarily be lost because those areas of a disk are usually reconstructable. What follows is a brief description of how to deal with particular kinds of data loss.

A disk can the divided into six basic kinds of data storage, which are listed below. DOS and ProDOS have all six kinds, while Pascal and CP/M do not have all six kinds.

1. The Operating System Boot Image
2. The Allocation Bitmap
3. The Directory
4. File Allocation Data (Track/Sector List, Index Block)
5. File data (programs, documents, data)
6. Unused sectors or blocks.

The first three are all generally reconstructable, although some information in the directory may be lost. Fortunately, the lost information is often not critical. The operating system boot image can be restored in several different ways, depending on which operating system the disk uses. The bitmap and directory of DOS or ProDOS disks can be reconstructed by FIXCAT.

The fourth and fifth areas of the disk, concerning files, are often impossible to reconstruct. The degree of difficulty depends entirely on the particular disk (how many files it contained, etc.) and on your knowledge concerning the missing data. For example, it might be easy to reconstruct a Track/Sector List on a relatively empty DOS disk, but the task may prove impossible on a disk that is full and has many files on it. FIXCAT can provide a great deal of helpful information by locating all valid file information. If the error is in the data area of the file, you are really at the mercy of how much you know and can remember about the file.

The sixth area, unused sectors, poses little threat, but it is best to repair these areas to insure no future problems. INIT works very well to recover those sectors.

In any case, if a portion of the disk is damaged you probably won't be able to write to the disk. If the disk in question is a 5 1/4" floppy, INIT can help you create a disk you can write to by reformatting the damaged sectors without destroying the rest of the disk. You're probably better off, though, to first make a copy of the damaged disk.

Whenever attempting to recover data from a damaged disk, it is best to make a copy of the disk to avoid further loss of data. This usually presents a problem, because most copy programs will not copy a disk with one or more bad sectors. We have provided the COPY function of the INIT program specifically to allow you to copy damaged diskettes.

Several tutorials in this documentation illustrate working with damaged diskettes. Three of these are "A FIXCAT tutorial" in Chapter 5, "Identifying and Correcting Format Errors" in Chapter 6, and "Locating and Fixing an I/O Error" in Chapter 6.

Order Form

ISBN/PRODUCT NUMBER	QUANTITY	DESCRIPTION	MEDIUM (CASS/DISK)	AMOUNT
			SUBTOTAL	
			(CA RESIDENTS) SALES TAX	
			SHIPPING	
			TOTAL	

CHECK # _____
OR VISA/MASTERCARD # _____ EXPIRES _____

NAME _____
STREET ADDRESS _____
CITY, STATE, POSTAL CODE _____
COUNTRY _____

Prices Subject to Change Without Notice

Ordering Directly from Quality Software

To order our products directly, mail the order form on this page to Quality Software (at the below address) with your payment—the price of the software (plus sales tax for California residents) *plus* shipping charges.* Your payment can be a check or bank draft made payable to Quality Software in US dollars, or your VISA or MASTERCARD number and expiration date (VISA and MASTERCARD holders may phone in their orders). California residents must add the appropriate sales tax (6 or 6.5%). No CODs will be accepted.

***Shipping Charges**

48 Continental United States (UPS)	$ 2.50
Alaska, Hawaii, Canada, and Mexico (air mail)	$ 5.00
All other countries (insured air mail)	$10.00

Send your orders to:

QS QUALITY SOFTWARE
21601 Marilla Street
Chatsworth, CA 91311
Phone (818) 709-1721

NOTES

NOTES

For Apple II, II+, IIe, IIc

Requires 64K RAM and one 5¼" disk drive

Bag of Tricks 2™

By Don Worth and Pieter Lechner

BAG OF TRICKS 2 is a set of four comprehensive disk utility programs that help analyze, modify, and restore disk data. Designed for 5¼" floppy disks, many of the features of BAG OF TRICKS 2 also apply to 3½" floppy disks, hard disks, and RAM disks.

BAG OF TRICKS 2 works with all disk operating systems used on the Apple II computer (DOS 3.2, DOS 3.3, ProDOS, Pascal, and CP/M). It is not copy protected, and thus can be backed up or transferred to any ProDOS volume.

BAG OF TRICKS 2 is useful to beginners and experienced programmers alike. It includes many "hand holding" tutorials that assist you in repairing damaged diskettes and allow you to change sector ordering, reconstruct damaged DOS catalogs or ProDOS directories, etc. etc. etc. It is one of the best software values ever offered for the Apple II computer.

The four programs and their functions are:

1. TRAX dumps and examines a track from a 5¼" floppy disk, displays the internal Apple diskette formatting information, and flags exceptions to standard formats.
2. INIT will copy the undamaged parts of one disk to another, even if parts of the original disk are damaged. It will reformat individual tracks of a 5¼" floppy disk, preserving the contents of undamaged sectors. It also allows you to change sector order.
3. ZAP is a sector editor like no other! More than 50 commands are available to assist you to locate, compare, change, or print the data on your diskettes. Design your own macro commands. Examine and modify DOS files, ProDOS files, Pascal files, even CP/M files on floppies, RAM disks, and some hard disks.
4. FIXCAT automates the process of repairing a damaged DOS catalog or ProDOS directory or subdirectory. It operates with or without user intervention, locating "lost" files and rebuilding the directory – from scratch if necessary! If it's possible to recover your directory, FIXCAT will do it.

QS QUALITY SOFTWARE
Computer Book Division

Bag of Tricks 2™

ISBN 0-912985-35-6 > $49.95US

PRINTED IN USA