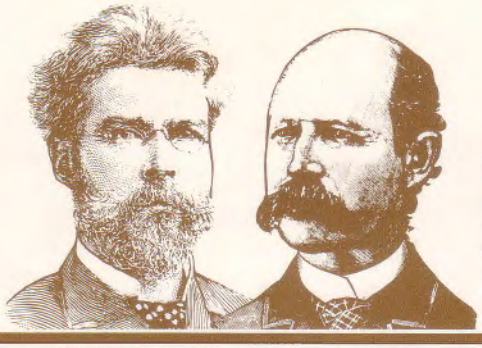


UNLOCKED and UNPROTECTED

Compatible with
- **Apple IIe**
- **Apple II+**
- **Apple II**

Beagle Bros™
Micro Software Inc.

APPLE* MECHANIC

Shape Creator / Byte Zap / Multiple-Utility Disk
by BERT KERSEY

SHAPE TABLE EDITOR
Put professional hi-res animation in your programs. Key-draw any shape on the screen, then let your Apple convert it into a shape table. All shapes are easily manipulated by Applesoft commands or by the programs on this disk.

CHARACTER EDITOR
Create custom typefaces, large or small. Make any character be any picture or symbol you want.

Proportionally-Spaced shape table type is easy to use and positionable *anywhere* on either hi-res screen (no vtab/htab restrictions). Included on the disk are six 96-character rotatable fonts, ready to use in titles or animated graphic presentations.

XTYPER and HI-WRITER
Type directly onto charts, graphs and hi-res pictures using the type styles, sizes & colors that you want. Even create "live" graphic presentations.

Includes three **LIST-able demos** that illustrate the many graphic techniques possible using ordinary Applesoft BASIC commands.

Note: The hi-res type routines and fonts on this disk are usable in your programs **without licensing fee**. Simply credit Beagle Bros on your disk and documentation.

BYTE ZAP
A super disk zapper that lets you rewrite any byte on a disk, and inspect and edit any track or sector. Uses Hex, Decimal *or* Ascii display and data entry. Create custom catalog titles, alter DOS, restore deleted files, make files inaccessible...

BONUS SUBROUTINES
Music and sound tricks, special screen wipes, hi-res picture movers. Plus...

APPLE TIP BOOK #5
Useful tips & tricks for your Apple—"More Room for Hi-Res Programs", "Hi-Res Flicks", "560-Plot Hi-Res"...

Also "Memory Movers", "Error Finders"... Hours of entertaining educational Apple reading material.

Free PEEKS & POKES
11" x 17" Wall Chart enclosed— Apple's PEEKS, POKES, POINTERS and CALLS on one heavy-duty poster. An indispensable programming tool.

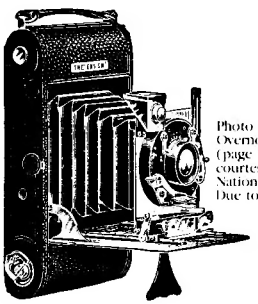
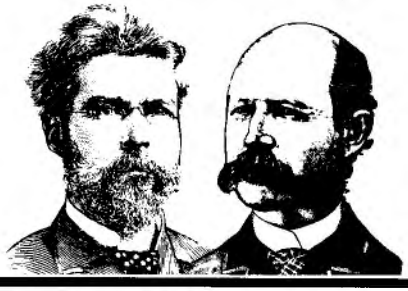


Photo credits: G.Bell (page 2), R.F.Modulator (page 3), Jose Canyussee (page 5, cover), Roger Overmout (page 7), Bill Overdoo (page 8, 13, 14), Mandy Lifeboats (page 9, 22, 58), Tom P.Ping (page 10), Hi-Res Harry (page 17) courtesy Fly-by-Night Circuses Intl., Apple-O Computer (page 15) courtesy Beagle Bros Hardware Research Department. Figure-8 illustration (page 26) courtesy National Museum of Overpriced Art. Free Cash offer (page 66) expires December 31, 1999.
Due to the spiraling cost of black ink, the top of this page was intentionally left blank.

Beagle Bros presents

Apple Tip Book #5

A New Collection of
APPLE II TIPS & TRICKS



Beagle Bros™ MICRO SOFTWARE

Plus Complete Instructions for Using

Apple Mechanic

SHAPE CREATOR/MULTIPLE-UTILITY DISK

by Bert Kersey

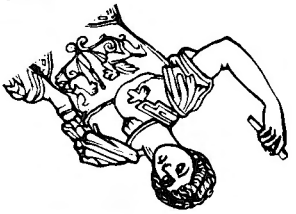
APPLE TIP BOOK #5

ERROR TOKENS

Tokens are machine language equivalents for the commands and characters in your Applesoft programs. See our *Tip Book #3* for more details. The tokens I don't understand are 235-255 (that's why you won't find them in *Tip Book #3*). Type this program and run it—

```
5  REM
6  FOR X = 235 TO 255: POKE 2053,X
7  LIST 5: NEXT : POKE 2053,178
```

Line 6 changes Line 5 and Line 7 lists it. But why all those *error messages?* Who knows?



NOT A PROGRAM YOU'LL USE EVERY DAY

```
10  TEXT : HOME : PRINT "ARABIC TO ROMA
    N CONVERTER (1-9999)"
20 X$ = "IVXCDMXLCIVX": GOTO 130
30  PRINT : INPUT "ARABIC NUMBER:";N$
40 N =  VAL (N$):L =  LEN (N$): IF N <
    1 OR N > 9999 THEN 30
50  IF L < 4 THEN  FOR I = 1 TO 4 - L:N
    $ = "0" + N$: NEXT
60  FOR I = 1 TO 4:N(I) =  VAL ( MID$ (
    N$,I,1)): NEXT
70  PRINT : PRINT "ROMAN NUMERAL:";
80  FOR I = 1 TO 4: RESTORE : FOR J = 1
     TO N(I) + 1: READ A$: NEXT : IF A
    $ = "0" THEN 120: DATA 0,1,11,111,
    12,2,21,211,2111,13
90  IF I = 1 AND N(I) < 4 THEN  FOR X =
    1 TO N(I): PRINT "M";: NEXT : GOTO
    120
100  FOR X = 1 TO  LEN (A$): PRINT  MID$
    (X$,3 * (I - 1) +  VAL ( MID$ (A$,
    X,1)),1);: NEXT
110  IF I = 1 AND N(I) > 3 THEN L =  LEN
    (A$): CALL  - 998: HTAB  PEEK (36)
     + 1 - L: FOR X = 1 TO L: PRINT  CHR$
    (95);: NEXT : CALL  - 922
120  NEXT
130  PRINT : PRINT : FOR I = 1 TO 40: PRINT
    "-";: NEXT : GOTO 30
```

FLASH FLASH FLASH

It's nice to be able to call attention to a screen statement, but Apple's FLASH is a bit too flashy for me. Instead of making it go *inverse/normal/inverse/normal,* I wish they had made it go *normal/nothing/normal/nothing.* That's why I sometimes use—

```
10  TEXT : HOME :Q$ =  CHR$ (34)
20  VTAB 9: HTAB 1
30  PRINT "PRESS      KEY."
40  FOR X = 1 TO 250: NEXT
50  VTAB 9: HTAB 1
60  PRINT "PRESS ANY KEY."
70  FOR X = 1 TO 250: NEXT
80 K =  PEEK ( - 16384): IF K < 128 THEN
    20
90  POKE  - 16368,0: VTAB 9: HTAB 1
100  PRINT "THANKS FOR PRESSING THE ";Q
    $; CHR$ (K - 128);Q$;" KEY."
```

DISK FIXER

If you've got a disk that won't boot or is giving you a bad time in some other way, there's just a *chance* that you can repair it with the *Master Create* program on your System Master disk. Just BRUN MASTER CREATE and insert the disk you want to fix. It will rewrite Tracks 0-2 (DOS) for you, and *maybe* that's where your problem was. While you're at it, *Master Create* will let you change the name of your disk's greeting program too. But then so will *Byte Zap,* and with a lot more flair!

NOTRACE CATALOG

You know how you can move the cursor up into a catalog with esc-I-I-I, etc., type "RUN", and then trace over the file name? Well, you *can't* do it if you run this program first:

```
10  PRINT  CHR$ (4);"CATALOG"
20  FOR I = 1 TO 47 STEP 2
30 S =  SCRN( 7,I): IF S = 12 OR S = 13
      THEN  COLOR= S - 4: PLOT 7,I
40  NEXT
```

Line 30 changes the first character of each file name to a *control* character (on the screen only). And even though ctrl-characters can sometimes be seen, they can't be traced!

HOW BIG IS THAT PROGRAM?

One way to tell the size of a program is to save it and look in the catalog. Every sector, not counting the first one, is about 256 bytes in size. Divide the sector count by 4 and you have roughly the size of the program in K's. For example, a 48 sector file occupies about 12K or 12,000 bytes.

Another way to tell a program's size is to load it into memory and then subtract it's start address from it's end address, like so:

```
PRINT (PEEK(175) + PEEK(176)*256)
  - (PEEK(103) +  PEEK(104)*256)
```

This typing exercise will tell you the number of bytes your program occupies. The start address is usually always 2049, so you can use 2049 instead of the second pair of peeks.

FUTURE FANCY FILER?

Remember Kareem Abdul Murphy from Tip Book #1; the guy who can do a CONTROL-RESET with one hand? Well, he just came up with this terrific program he's wants to market. He says it still has some bugs, but will be finished near the end of next month (where have we heard that before?). Anyway, here's his program. Anyone out there know anything about software marketing?

```
12  HOME
19  VTAB 1: HTAB 1: INVERSE
26  PRINT " FILER ": NORMAL : PRINT
33  PRINT "<F>  FIND FILE"
40  PRINT "<F>  FILL FILE"
47  PRINT "<F>  FIX FILE"
54  PRINT "<F>  FACILITATE FILE"
61  PRINT "<F>  FOCUS FILE"
68  PRINT "<F>  FILE FILE"
75  PRINT "<F>  FINISH"
82  VTAB 15: PRINT "SELECT FUNCTION:";:
      GET F$: PRINT F$
89  VTAB 15: INVERSE : PRINT " NOT AVAI
     LABLE ";: CALL  - 958: NORMAL
96  NORMAL : FOR I = 1 TO 999: NEXT : GOTO
     19
```

FUNNY INVERSE

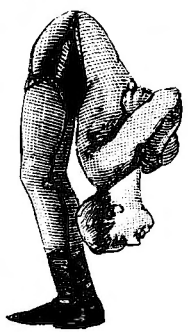
We just bought a new Apple and it puts out inverse type that looks like this:

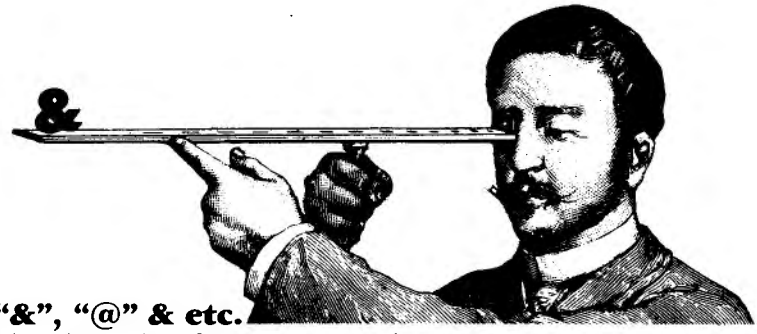
INVERSE

instead of our old Apple's style:

INVERSE

Nobody knows why. We all prefer the old version, probably because we're used to it. Can anyone out there tell us how to convert our hardware to the old style inverse?





"&", "@" & etc.

All right, lookit, if you're going to be serious about this computer stuff, it's time you got your character names straight. Here's the rundown; there will be a quiz on Monday—

& This is called an *ampersand.*
@ This is called an *at sign.*
* This is called an *asterisk*
] This is called a *square bracket.*
) This is called a *parenthesis*
^ This is called a *little pointy thing.*

WHO-CARES DEPARTMENT

"PRINT ." (return) will print a zero. "PRINT . ." will print *two* zeros. "LET N = ." sets N equal to zero. "LET N = . ." hits you with a ?SYNTAX ERROR.

NUMBER UNCRUNCHER

Have you ever wanted to dissect a decimal number into its separate digits? Here are two ways to do it:

```
10  INPUT "ANY WHOLE NUMBER:";N:N =  INT
    (N):L =  LEN ( STR$ (N)): IF L > 9
     THEN 10
20  FOR X = 1 TO L:E = L + 1 - X:MOD =
    INT (N / 10 ^ E) * 10 ^ E:MOD = N
    - MOD:D(X) =  INT (MOD / (10 ^ (E
    - 1))): NEXT
30  FOR X = 1 TO L: PRINT "DIGIT #";X;"
    : ";D(X): NEXT
40  PRINT : GOTO 10
```

The above method does the dissecting mathematically. Below, we do it with strings and things. It is much *much* faster—

```
10  INPUT "ANY WHOLE NUMBER:";N:N =  INT
    (N):N$ =  STR$ (N):L =  LEN (N$): IF
    L > 9 THEN 10
20  FOR X = 1 TO L:D(X) =  VAL ( MID$ (
    N$,X,1)): NEXT
30  FOR X = 1 TO L: PRINT "DIGIT #";X;"
    : ";D(X): NEXT
40  PRINT : GOTO 10
```

ONERR ERROR LISTER

Here's a good one— This program has an intentional error in Line 12345. Type it in *exactly* as you see it. Be careful; a typo in Line 60000 can really create a big mess when you run the program (I know; I've been there).

```
10  ONERR  GOTO 60000
1000  REM  YOUR PROGRAM HERE
12345  PRINT :PIRNT: PRINT : REM TEST;
     2ND "PRINT" MISSPELLED
59999  END
60000 T = 256:E =  PEEK (220) + T *  PEEK
     (221):E = E + 5 * ( PEEK (E) = 0):
     V =  PEEK (E): POKE E,207:L$ =  RIGHT$
     ("0000" +  STR$ ( PEEK (218) + T *
      PEEK (219)),5):L =  PEEK (121) +
     T *  PEEK (122) + 49: FOR I = 1 TO
     5: POKE L + I, ASC ( MID$ (L$,I,1)
     ): NEXT : LIST 00000: POKE E,V: POKE
     216,0: RESUME
```

Save the program and then run it. What you will get is the usual error message *and* a listing of the offending line with an arrow (greater-than sign) pointing to the statement that caused the error! No more searching; a *big* help if you're like me and use six pounds of statements on each line. Add Lines 10 and 60000 (could be any line numbers) to one of your programs in progress. It's a great de-bugger!

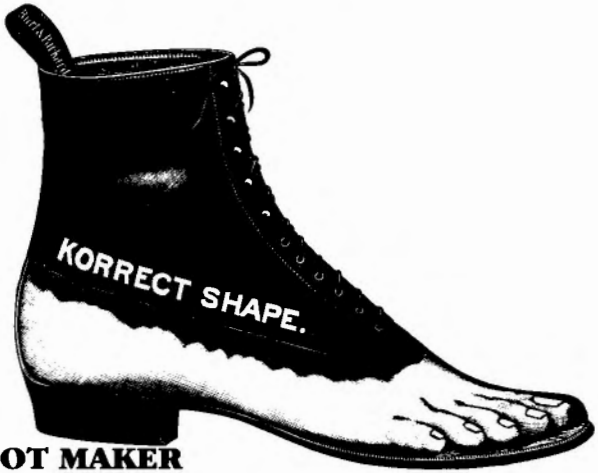
LEGAL NAMES DEPARTMENT:

VARIABLES must start with an upper case alphabetical character A-Z. The second character may be A-Z or a digit 0-9 (or nothing). Characters after that don't count but they have to be letters or numbers. FILE NAMES, however, may start with your choice of 64 normal or control characters (not to mention inverse and flashing), but *not* characters with ASCII values 32-63. File names may not contain commas.

BLACK + WHITE = COLOR

You can't trust anything. Here's a program that makes color lines out of black and white. I'd explain it, but it's 2 a.m. and I'm at a loss for words right now.

```
10  HOME : HGR : HCOLOR= 3: REM (OR 7)
20  FOR Y = 10 TO 40 STEP 10: HPLOT 0,Y
      TO 279,Y: NEXT : REM (PLOT WHITE
     LINES)
30  FOR X = 0 TO 200 STEP 2
40  HCOLOR= 0: HPLOT X,10: HPLOT X + 1,
     20: REM  (ADD #0-BLACK TO MAKE #1-
     GREEN & #2-VIOLET)
50  HCOLOR= 4: HPLOT X,30: HPLOT X + 1,
     40: REM  (ADD #4-BLACK TO MAKE #5-
     ORANGE & #6-BLUE)
60  NEXT X: VTAB 20: END
```



BOOT MAKER

Did you ever want to make someone boot your disk before they could run your program? Here is a simple way to do it:

1. Put a blank disk in Drive 1.
2. Type "POKE 47721,123" (return).
3. Type "NEW" (return).
4. Type "INIT HELLO" (return).

In a minute or so, you will have a new master disk from which you will make copies. Use LOAD and SAVE (or *FID*) to transfer your programs to the master disk. Now put this command near the start of each of your programs:

```
1  IF  PEEK (47721) <  > 123 THEN  PRINT
      CHR$ (4);"PR#"; PEEK (43626)
```

47721 is a location unused by DOS 3.3. PEEK(43626) is the most recently used slot number. "123" could be any number, 0-255, as long as it matches the poke number in step 2. Odds are 255 to 1 that your disk will be booted before your program is run.

THE INCREDIBLE SHRINKING NUMBERS

This program demonstrates Apple's handling of small numbers:

```
10 X = 1
20 X = X / 10: PRINT X: GOTO 20
```

And *big* numbers:

```
10 X = 1
20 X = X * 10: PRINT X: GOTO 20
```

I don't know how they came up with the "E". Probably stands for **E**xtrahuge and **E**xtrateensy, right?

AHA! A USE FOR VERIFY

When you want to READ a text file, you first have to OPEN the file. If the file doesn't exist, you'll get an "END OF DATA" message *and* you'll acquire a new one-sector file name in your catalog, even if you used ONERR to trap the error. Fooey! To prevent this, VERIFY the file first, *then* OPEN it. If VERIFY doesn't find the file, ONERR catches your mistake, and life goes on with no file added to your catalog.

MECHANICAL MEMORY MOVER

Have you ever wanted to move a chunk of memory from within a program without going into machine language or using Applesoft's clunky CALL—468? Well, here's an idea; it may be a little slow, but it sure works. Just BSAVE the memory chunk and then BLOAD it where you want it. For example, to store the text screen ($400-7FF) at A$6000, these lines could be in your program—

```
1000  PRINT  CHR$ (4);"BSAVE TEXT,A$400
      ,L$3FF"
1010  PRINT  CHR$ (4);"BLOAD TEXT,A$6000"
```

The following will clear the screen, pause, and then bring the text screen back where it belongs—

```
2000  HOME : FOR W = 1 TO 2345: NEXT
2010  PRINT  CHR$ (4);"BLOAD TEXT"
```

FREE APPLE MAGAZINE

If you have never received *Softalk* Magazine, a really good (and big) Apple publication, you can get it **FREE** for a year, by sending your Apple's serial number with a request for subscription to—

SOFTALK Circulation
11021 Magnolia Boulevard
North Hollywood, Ca 91601

GRAND PRIX APPLE

Peter Lake, from somewhere-out-there, called us here at the Beagle Building to report on his customized Apple environment. It seems Peter is a professional writer and uses an Apple as a word-processor all day, every day. So he mounted his system on a drawing board tilted about 15 degrees. To level his monitor, he built a wedge-shaped mount that rests on his Apple. He painted his Apple's case black to reduce glare (I never noticed the problem) and, inspired by one of our tips, glued *sequins* to his K and D keys so he can feel his fingers' position on the keys without looking. Now the best part: He sits in a special *race car seat*, head rest and all, leans back and types away. I asked him if he wears a seat belt; he said, "Only when I type fast."

LC EQUIVALENTS:

If you have lower-case hardware in your Apple, you can enjoy lower-case characters on your text screen. If you write programs for other people's Apples, however, you don't *dare* use lower-case text-screen text (hi-res lower case is no problem). You must assume that no one else has lower case capabilities. To see the equivalent characters that others might see when a program says PRINT "Hello", type:

```
1Ø  INVERSE : PRINT "Hello"
```

In this case, you will be presented with H%,,/ (in INVERSE). That's what non-lower-case Apples will produce, even in NORMAL mode. By the way, this little tip helps prove that there's *no way* to print INVERSE or FLASHing lower case.

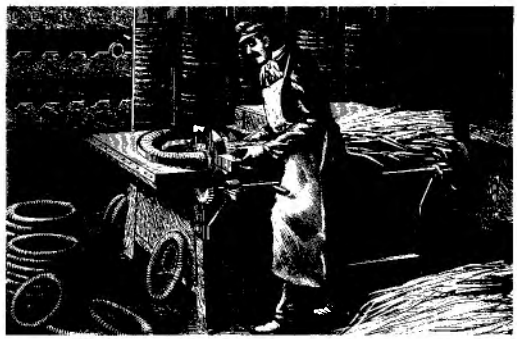
If you are about to add lower case hardware to your Apple, shop around. Have your dealer *show you* a comparison of the typestyles available. A couple of the popular ones look like they were designed by engineers, not typographers. Poorly designed (ugly) text screen type can really slow you down, especially if you are doing word processing.

HOW MUCH DATA?

If you wonder how many items you have in a DATA statement, don't count them; let your Apple count them. Temporarily insert a giant loop at the start of your program, like:

```
1  FOR X = 1 TO 1ØØØØØØØ: READ A$: NEXT
```

Run it. Sooner or later, you'll get an "?OUT OF DATA" error message. Now type "PRINT X—1" (return). The value you get for X—1 will tell you the number of items of DATA you have.



DELAY LOOPS

When you want a program to stop and wait, make your Apple sit there and count to some big number (your Apple doesn't mind; in fact it's *good* for him). Insert a statement like—

```
1ØØ  FOR X = 1 TO 2ØØØ: NEXT X
```

Every 1000 is approximately a one-second wait. If your program is going to pause many times, use a delay loop as a subroutine (GOSUB) to save memory.

COLUMN POKER

To make a flush left column in the middle of the screen, you can do this:

```
10  HOME : VTAB 10: HTAB 15: PRINT "A STI
    TCH": HTAB 15: PRINT "IN TIME": HTAB
    15: PRINT "IS WORTH": HTAB 15: PRINT
    "TWO IN": HTAB 15: PRINT "THE BUSH."
```

But why not poke locations 32 and 33 and do this?

```
10  VTAB 10: HTAB 15:M$ =  CHR$ (13): POKE
    32,14: POKE 33,26: PRINT "A STITCH"M
    $"IN TIME"M$"IS WORTH"M$"TWO IN"M$"T
    HE BUSH.": TEXT
```

The second method takes less typing time and less memory. Just poke 32 with a number *one less* than the column number, and poke 33 with 40 *minus* PEEK(32). You can sometimes forget the poke at 33 if you are *not* going to print off of the screen (causing wrap-around). If you *do*, and PEEK(32) plus PEEK(33) totals more than 40, you can—

(a) zap your program.
(b) make your drive go "BLAA-AATT!"
(c) cause wrap-around to the wrong line.
(d) prove me wrong.

A DIFFERENT APPROACH

If a programming solution is getting you down, try looking at it from a whole new angle. And if you're *really* stuck, start over from scratch.

ERRORLESS ERRORS?

Applesoft just loves to crash when you specify a number it can't handle. Try HPLOT 0,300 for example. You'll get an ?ILLEGAL QUANTITY message because 300 is too large a number for a hi-res plot. Wouldn't it be nice sometimes if Applesoft just ignored an error instead of scolding you? Well it seems to do just that with its MID$, LEFT$ and RIGHT$ commands. Try this:

```
10 A$ = "FISH": FOR X = 1 TO 9
20  PRINT X;".";  LEFT$ (A$,X): NEXT
```

Theoretically the program should bomb when X becomes greater than the length of A$ which is 4. But it doesn't! For example, when X is 7, Line 30 asks the computer to print the *left 7 letters* of a 4-letter word. MID$ and RIGHT$ seem to work the same way, and I don't mind at all; it's one detail I *don't* have to look for when debugging.

```
DECIMAL DISASSEMBLER
   BY BERT KERSEY
   BEAGLE BROS
```

(Read my column in the April 82 Softalk for more details.)

```
20  TEXT : HOME : NORMAL
30  DIM SC(24): FOR I = 1 TO 24:SC(I) =
      128 * I + 1 - (984 *  INT ((I - 1)
      / 8)) + 895: NEXT
40  GOTO 120
50  POKE 58,ST -  INT (ST / 256) * 256:
      POKE 59, INT (ST / 256): REM (STO
    RE START LOCATIONS)
60  HOME : INVERSE : PRINT "---------HEX-
    ------";: HTAB 21: PRINT "-------DEC
    IMAL------";: NORMAL
70  VTAB 1: CALL 65121: REM (DISASSEMBLER)
80  FOR V = 2 TO 21:H = 0: GOSUB 150:XH
    I = 256 * X:H = 2: GOSUB 150: VTAB
    V: HTAB 21: PRINT XHI + X;"- ";: CALL
     - 868: REM (PRINT DEC.LOCATIONS)
90  HTAB 28: FOR H = 8 TO 14 STEP 3: GOSUB
    150: IF X >  = 0 THEN  PRINT  SPC(
    (X < 100) + (X < 10));X;" ";
100  NEXT : NEXT
110  PRINT : VTAB 24: PRINT "OR <RETURN
    > TO CONTINUE OR <Q> TO QUIT.";
120  VTAB 23: HTAB 1: INPUT "ENTER NEW
    START LOCATION:";ST$: IF ST$="" THEN 60
130  IF ST$ = "Q" THEN  VTAB 22: CALL  -958
    : END
140 ST =  VAL (ST$): GOTO 50
150 P1 =  PEEK (SC(V) + H):P2 =  PEEK (
    SC(V) + H + 1):X = 16 * (P1 - 176 -
    7 * (P1 > 185)) + P2 - 176 - 7 * (
    P2 > 185): RETURN
```

MEM SAVER

To draw a line of hyphens on the text screen, you can use:

```
1Ø  PRINT "----------------------------------
    ----------"
```

This works fine of course, but it uses 43 bytes of memory every time you do it. No problem if you've got memory to burn. The following line does the same job and uses just 15 bytes:

```
1Ø  FOR X = 1 TO 4Ø: PRINT "-";: NEXT
```

Remember, if you need to draw this line several times in the same program, set it up as a subroutine. Just tack a ": RETURN" on the end. The cost? Only two more bytes; one for the colon and one for the "RETURN".

GET INFO

GET is a funny animal. Here are today's GET quirks:

a. If you GET a number with a statement like "GET X", but input a letter, the program bombs with a ?Syntax Error.

b. With P.L.E., GET interprets a Right Arrow (ctrl-U, ascii 21) as a *space* or whatever character happens to be under the cursor, *but* it interprets a Left Arrow (ctrl-H, ascii 8) correctly.

c. GET interprets the length of a carriage return as 1, whereas INPUT considers the length zero.

d. A GET A$ won't let ctrl-C exit a program unless you say "IF A$=CHR$(3) THEN END".

Here is a getter that might teach you something:

```
1Ø  PRINT "ANY KEY:";: GET A$
2Ø  PRINT A$, ASC (A$): GOTO 1Ø
```

Run this program and try out the above quirks.

PLEASE ANSWER (Y/N):

If your program is looking for a yes or no answer, it is best to GET a one-character answer, something like:

```
1Ø  PRINT "ANSWER (Y/N):";: GET A$
```

Oh, yes— something we learned the hard way: Someone could answer with a *lower case* "y" or "n". I think Murphy's Law says you should be ready for this. GET, by the way, is better than:

```
1Ø  INPUT "ANSWER (YES/NO):";A$
```

The problem with INPUT is that (I don't know *why*) people often put a *space* after their answer, and even the dumbest Apple knows that "YES" is not equal to "YES ".

STEP-LIST

During a long listing, hold the CTRL key and the S key with one hand and the REPT key with the other hand. You can slow your listing down this way.

Note: The above tip has been classified as *worthless* by the Beagle Bros staff. Please tear it out of your book right away. Thank you.

SEMI-SUPER SORTER

This little program writes fifty useless (and hopefully clean) words for you and then alphabetizes them in around 15 seconds. Can you write an Applesoft program that will do the sort faster? On your marks...

```
10 N = 50: REM NUMBER OF WORDS
20  DIM A$(N)
30  HOME : PRINT "<";N;" RANDOM WORDS>"
    : PRINT
40  FOR I = 1 TO N: FOR J = 1 TO 5 +  INT
    ( RND (1) * 5):A$(I) = A$(I) +  CHR$
    ( INT ( RND (1) * 26) + 65): NEXT
    : PRINT I;".";A$(I): NEXT
50  PRINT : PRINT "<SORTED LIST>": PRINT
60 A = 1:L = 0:Z = 1
70  FOR X = A TO N: FOR Y = A TO N: IF
    A$(X) > A$(Y) THEN A = Y: GOTO 70
80  NEXT :L = L + Z:X$ = A$(X):A$(X) =
    A$(L):A$(L) = X$: PRINT L;".";X$:A
    = L + Z: IF A <  = N THEN 70
90  NEXT
```

WATCH YOUR 3's AND 2's

Don't fret if you have to look at the keys when you type numbers. Mistyped numbers are the source of some of the toughest programming mistakes to find. I've found it pays to slow your typing down about two beats when numbers come up, and it doesn't hurt to *watch what you're doing* either.

←POP GOSUB→

?POP WITHOUT GOSUB ERROR

POP is an Applesoft command to be used when you do a GOSUB but then decide not to RETURN. Without POP, things may work alright for a while, but sooner or later you're going to get an ?OUT OF MEMORY error, as in—

```
10  PRINT X;" SUIT";: GOSUB 50
50  PRINT "CASE ":X = X + 1
60  GOTO 10
```

When X reaches a value of 24, you'll get an error message. Line 60 *should* read—

```
60  POP : GOTO 10
```

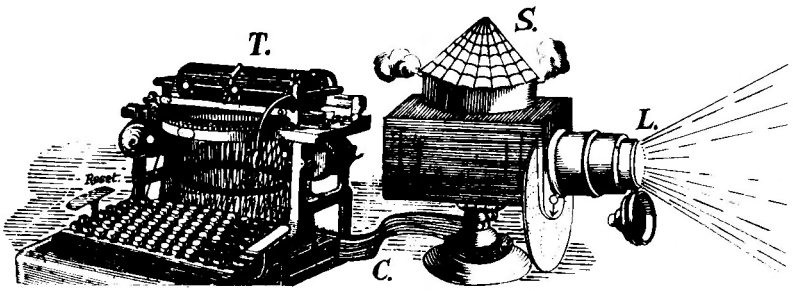
If a program encounters a POP when *no* GOSUB has been executed, you will get a ?RETURN WITHOUT GOSUB error message. Not quite correct, but you get the message.

ANOTHER FREE CHART!?

If you've got an 80-column printer, here's a handy chart program. Just type it in, run it, and get out of the way—

```
10  FOR I = 1 TO 69:L$ = L$ + "-": NEXT
15  PR# 1: REM PRINTER SLOT
20  FOR I = 1 TO 6: PRINT : NEXT
30  PRINT  SPC( 3);"INVERSE"; SPC( 8);"
    FLASH"; SPC( 10);"CONTROL"; SPC( 1
    4);"NORMAL": PRINT L$
40  FOR V = 0 TO 63: FOR N = V TO V + 1
    92 STEP 64
50  IF N > 159 AND N < 255 THEN X$ =  CHR$
    (N): GOTO 90
60  IF N = 255 THEN X$ = " ": GOTO 90: REM
    MAKES CHR$(255) PRINT AS A SPACE.
    DEPENDS ON YOUR PRINTER.
70 X = N:X = X + 192 * (N < 32) + 128 *
    (N > 31 AND N < 96) + 64 * (N > 95)
80 X$ =  CHR$ (X)
90  PRINT "   ";: GOSUB 140: PRINT " ";
    X$;" ";N; SPC( (N < 100) + (N < 10
    ));" ";: IF N > 127 THEN  PRINT "(
    ";N - 128;") "; SPC( (N - 128 < 10
    0) + (N - 128 < 10));
100  PRINT "  ";: NEXT N: PRINT : IF V +
    1 -  INT ((V + 1) / 16) * 16 > 0 THEN
    130
120  PRINT :T = T + 1: IF T = 2 THEN  PRINT
     SPC( 3);"INVERSE"; SPC( 8);"FLASH
    "; SPC( 10);"NORMAL"; SPC( 15);"LO
    WER CASE": PRINT L$
130  NEXT V: FOR I = 1 TO 6: PRINT : NEXT
    : PR# 0: END
140  PRINT "$";:N% = N / 16: GOSUB 150:
    N% = N - N% * 16: GOSUB 150: RETURN
150  PRINT  CHR$ (48 + N% + 7 * (N% > 9
    ));: RETURN
```

This chart shows you the values for all 256 normal, inverse, flashing and control characters. It also serves as a handy ASCII conversion chart and 0-255 hex converter. The program *was* going to include a complete biorhythm chart for everyone born in this century and a full-size hi-res rendering of Zasu Pitts, but we're saving those features for Tip Book #32767.



HI-RES FLIX

This program lets you flip between two hi-res pictures at a predetermined rate. Variable W1 controls the time page one shows, and W2 does the same for page two—

```
10  TEXT : HOME
20  INPUT "PAUSE 1 (0-500):";W1
30  INPUT "PAUSE 2 (0-500):";W2
40  POKE  - 16304,0: POKE  - 16302,0: POKE
      - 16297,0
50  POKE  - 16300,0: FOR I = 1 TO W1: NEXT
    : POKE  - 16299,0: FOR I = 1 TO W2
    : NEXT : GOTO 50
```

BLOAD any two hi-res pictures before you run the program by typing "BLOAD PICTURE-1,A$2000" (return) and "BLOAD PICTURE-2,A$4000" (return). $2000 & $4000 represent the addresses of page 1 & 2. "PICTURE-1" and "PICTURE-2" are the names of your 2 hi-res pictures. Run the program and notice the effect different rates of flicker have. If you want, remove the delay loops entirely from Line 50.

A machine-language version of the same program resides below. Notice how (humungously) much faster it is! It is *so* fast that if you use a small wait value (W1 or W2), a picture doesn't have time to be completely "printed" on the screen before the page switch is made. You'll have to stop the program with RESET.

```
10  TEXT : HOME
20  INPUT "PAUSE 1 (0-80):";W1
30  INPUT "PAUSE 2 (0-80):";W2: IF W2 >
      80 OR W1 > 80 THEN 10
40  POKE 768,141: POKE 769,84: POKE 770
    ,192
50  POKE 771,62: POKE 772,0: POKE 773,0

60  FOR I = 4 TO W1 + 3: POKE 770 + I,2
    34: NEXT
70  POKE 770 + I,141: POKE 771 + I,85: POKE
    772 + I,192
80  FOR J = 1 TO W2: POKE I + J + 772,2
    34: NEXT
90  POKE 772 + J + I,76: POKE 773 + J +
    I,0: POKE 774 + J + I,3
100  POKE  - 16304,0: POKE  - 16302,0: POKE
      - 16297,0
110  CALL 768
```

SUPER DEL

Sometimes you may want a program to delete its first few lines, especially if those lines BLOAD a file and you don't need it BLOADed the next time you run it. The DEL command will work in a program, BUT the program stops right there. So forget DEL. Instead, why not have your program change the start of program pointer? You can do this "on the fly", telling your Apple that the third line (or any line) of your program is now the *first* line (an Apple will believe anything). Type this program, SAVE it, RUN it, and finally LIST it. The first two lines, which could be BLOADers, will disappear! To get them back, do two pokes— POKE 103,1 and POKE 104,8.

```
10  REM THIS IS LINE 10.
20  REM THIS IS LINE 20.
30 LOC =  PEEK (121) +  PEEK (122) * 256
     + 1: POKE 103,LOC -  INT (LOC /
     256) * 256: POKE 104, INT (LOC / 256)
```

ERROR BREAK

If you try to load a non-existent file with an immediate (not in a program) command, "LOAD XYZZY", you get a normal "FILE NOT FOUND" message. If you have a program in memory that *has been run*, and type an immediate (again, *not* in your program) "*RUN* XYZZY", you get a "FILE NOT FOUND, BREAK IN 123" message, where 123 is the last line executed.

BOX EYES & SNAKE CARS

One reason I'm a computer nut is that I like messing with statistics. Which reminds me of this little dice rolling program that proves that sooner or later, 1 out of 6 dice rolls will come up totaling seven:

```
10  HOME : ONERR  GOTO 130
20 I = I + 1
30  VTAB 12: HTAB 1
40 SUM = SUM + 1:A =  INT (6 *  RND (1)
     ) + 1:B =  INT (6 *  RND (1)) + 1:
      NORMAL : IF A + B = 7 THEN SEV =
     SEV + 1
50  PRINT "  DICE ROLL:"; SPC( A < 10);
     A;" +"; SPC( B < 10);B;" = ";
60  IF A + B = 7 THEN  INVERSE : PRINT
     " 7 ";: GOTO 80
70  NORMAL : PRINT A + B;: CALL  - 868
80  PRINT : PRINT : NORMAL
90  PRINT  SPC( 5);"SEVENS:"; SPC( (SEV
     < 1000) + (SEV < 100) + (SEV < 10
     ));SEV
100  PRINT  SPC( 12);"-----  =";: HTAB 2
     3: PRINT ".0%";: HTAB 21: PRINT  INT
     (1000 * SEV / SUM) / 10
110  PRINT "TOTAL ROLLS:"; SPC( (SUM <
     1000) + (SUM < 100) + (SUM < 10));
     SUM
120  GOTO 20
130  END : REM JUMPS HERE IF CTRL-C
```

560-PLOT HI-RES

This program proves the existence of 560 horizontal hi-res plots. First, a normal bumpy hi-res line is drawn from 0,0 to 75,150 (DX,DY). Then we draw a parallel super-smooth (but slow) line.

```
10  HGR : HOME
20 DX = 75: DY = 150
30  HCOLOR= 3: HPLOT 0,0 TO DX,DY: REM
    NORMAL LINE
40  FOR Y = 0 TO DY
50 X = 20 + 2 * (DX * Y / DY)
60 ODD =  INT (X) -  INT (X / 2) * 2
70  HCOLOR= 3 + 4 * ODD
80 X = X / 2
90  HPLOT X,Y: REM OR X,Y TO X+1,Y
100  NEXT : VTAB 20: END
```

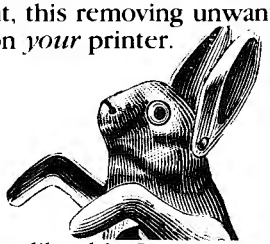
If the column number is *odd*, we use HCOLOR #7 white. If it's *even*, we use #3 white. These two colors plot in either the left or right half of one of the 280 hi-res columns. Our example here cheats just a little by picking special coordinates for extra smoothness.

AN 80-COLUMN LISTER

```
10 D$ =  CHR$ (4)
20  PRINT D$;"OPEN LIST80"
30  PRINT D$;"WRITE LIST80"
40  PRINT "PR#1"
50  PRINT "PRINT ";: CHR$ (34);: CHR$ (9)
    ;"80N";: CHR$ (34): REM PRINT "(ctrl
    -I)80N"
60  PRINT "LIST"
70  PRINT "PR#0"
80  PRINT D$;"CLOSE"
```

(I can't guarantee this program will work on your printer unless it's an Epson MX-70, 80 or 100.)

This program will create a text file called "LIST80". Now any program in memory will be listed on your printout in eighty columns (or the number specified in Line 50), by typing **EXEC LIST80**. The same effect may be created by typing PR# I (printer slot), and PRINT "(ctrl-I)80N". This command turns off the video output, this removing unwanted carriage returns. I hope it works on *your* printer.



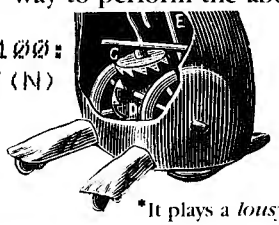
BUGS

I really hate bugs like this; I mean programming is hard enough without Applesoft's casual attitude with numbers. What answer does your Apple give for this?

```
PRINT INT(14.4 * 100)
```

1439, right? Heck, my eight-dollar pocket *calculator* does better than that.* A better way to perform the above calculation seems to be:

```
N = 14.4 * 100:
   PRINT INT(N)
```



*It plays a *lousy* game of *Raster Blaster* though.

SCREEN FORMULA

I've been trying to come up with this one for years (or at least since last week)— It's the formula for converting HTAB and VTAB coordinates into the corresponding memory address. Let V equal the VTAB (1-24) and H equal the HTAB (1-40). To find the memory location, LOC, for the character in that screen position, use (are you ready?):

```
LOC=128*V+H-(984*INT((V-1)/8))+895
```

Hey, it works! Now, to find what character is residing at VTAB V, HTAB H, you can peek at LOC and get its value, like so:

```
PRINT PEEK(LOC)
```

Or, you can poke any character you want onto the screen with:

```
POKE LOC, X
```

(For character values, see *Free Chart*, page 14.)

A PRINT-USING GOSUB

Applesoft loves to abbreviate numbers, quite a pain in the chips when you're working with dollars & cents. $3.20 will print as "$3.2". $3.00 will print as "$3". And to make matters worse, $3.2754 will print as "$3.2754". "PRINT-USING" is a useful command found on other computers (I *think;* I just found out that there *are* other computers). This command lets you specify ahead of time the decimal format for your screen and printer printouts. Well, the best I can do for you is this little GOSUB that rounds numbers to the nearest cent. Position the cursor where you want it (for example VTAB 20: HTAB 19), set the variable AMT equal to the number you want printed, and GOSUB 9999. The number will be rounded to the nearest penny and printed for you in dollars and cents format.

```
10 AMT = 12: GOSUB 9999
20 AMT = 12.3: GOSUB 9999
30 AMT = 12.34: GOSUB 9999
40 AMT = 12.345: GOSUB 9999
50 AMT = 123.456: GOSUB 9999
55 AMT = 1234.567: GOSUB 9999
60  PRINT "----------"
70 AMT = SUM: GOSUB 9999
100  END
9999 AMT = 100 * (AMT + .005):AMT =  INT
     (AMT): REM ROUNDS.OFF
10000  PRINT "$";: SPC( (AMT < 100000) +
     (AMT < 10000) + (AMT < 1000));AMT /
     100;: REM ALIGNS
10020  IF  INT (AMT -  INT (AMT / 100) *
     100) = 0 THEN  PRINT ".00";: GOTO
     10040
10030  IF  INT (AMT -  INT (AMT / 10) *
     10) = 0 THEN  PRINT "0";
10040 AMT = AMT / 100: PRINT :SUM = SUM
      + AMT: RETURN
```

Apple Mechanic Instructions

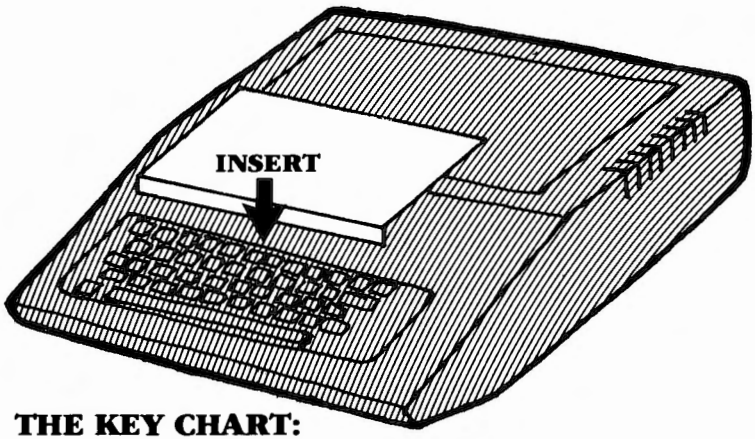
Welcome to APPLE MECHANIC! There is a large variety of programs on this disk, and as always, we hope you will find them both useful *and* entertaining. In using this disk, you will hopefully learn some new Apple programming techniques that will help you write better programs. With this in mind, all *Apple Mechanic* programs have remained listable and inspectable by normal methods. Let's get on with the instructions—

BACK UP YOUR DISK:

Unlike most commercially-made disks, *Apple Mechanic* is not copy protected (just copy*right*ed) and supports normal Apple DOS. Either make a back-up with one of the copy programs on your *System Master* disk or use the *FID* program from the same disk. And store the original in a safe place, away from prying magnetic fields.

BOOTING:

Booting will not be necessary if you have recently booted a normal-DOS disk and haven't been doing any exotic fooling around. If a program on *Apple Mechanic* gives you a problem, try typing "FP" (return) and re-running the program, or go ahead and boot. For you newcomers out there, *boot* means turn your Apple off, insert your disk in Drive 1, close the door and turn the Apple on. Or leave the power on and type (usually) "PR#6" (return). When your drive's red light goes off, you and your Apple are ready to go to work.



THE KEY CHART:

A two-sided Key Chart has been included with your disk. FOLD IT LENGTHWISE with the appropriate side facing up, and insert it behind the top row of keys on your Apple. The Key Chart provides convenient labels for the "key" keys in several of the *Apple Mechanic* programs and puts the instructions in front of you where they will do the most good. Functions that are not specified on the Key Chart will appear on your monitor at the appropriate times. Fold and store your Key Chart with your disk. (Note: Leave your disk *un*folded; it'll work much better that way.)

The Catalog:

APPLE MECHANIC
A simple "Hello" program that runs when you boot the disk.

SHAPE EDITOR (page 24)
Allows you to create 12-shape shape tables for use in your programs.

FONT EDITOR (page 31)
Allows you to edit and create hi-res character sets or "shape-fonts" to be used with your programs, *Xtyper* or *Hi-Writer*.

FONT SPLITTER (page 42)
Allows you to reduce the number of characters in any shape-font.

SHAPE ANALYZER (page 43)
Lets you analyze shape tables and test each shape in various combinations of ROT, SCALE and HCOLOR.

XTYPER (page 34)
Lets you type shape-font characters directly onto a hi-res image or blank screen. The finished image may then be saved to disk.

HI-WRITER (page 38)
Allows program-control of shape-fonts. You add your own Applesoft commands to make "live" graphic presentations.

HI-WRITER DEMO (page 38)
List this to see how to make your own graphic presentations.

SHAPE TABLE DEMO (page 27)
A demonstration of shape table commands and loading procedures.

GREETINGS
Our "talking heads" say hello, using most of the graphics and sound features on *Apple Mechanic* (and then some).

PAGE COPY (page 42)
Three EXEC programs that move images between pages 1, 2 & 3.

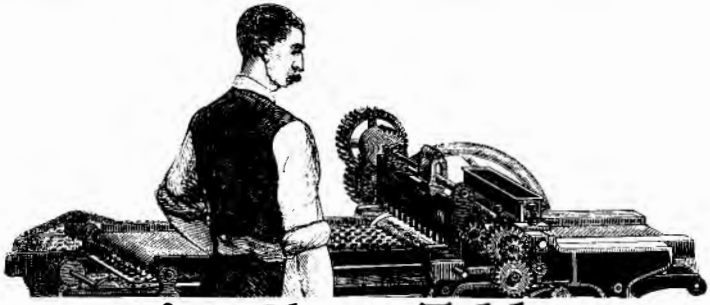
BYTE ZAP (page 49)
A handy disk detective that lets you rewrite any byte on a disk.

SONG SUBROUTINES (instructions on disk)
A few little tunes and noises to liven up your programs.

TEXT TRICKS (instructions on disk)
Some text screen gimmicks; we couldn't resist.

TWO-LINERS
Spectacular winning entries from Uncle Louie's famous contest.

SHAPES, @IMAGES AND]FONTS
These are binary picture files and shape tables. *Cursors* and *Demo Shapes* are shape tables used by the editing and demo programs. Hi-res image file names are preceded by an "@". File names that start with a "]" (shift-M) are shape-fonts. You may, of course, use your own system for naming your own files.



Creating Shape Tables

Shape Tables are excellent hi-res animation tools. Each "table" is simply a series of hi-res drawings stored in memory in the form of plotting and non-plotting vectors. Shapes in each shape table may be called to the screen by Applesoft's DRAW and XDRAW commands. While being rather abstract in concept, shape tables are relatively easy to use. Difficulty often comes in *writing* shape tables. One look at your Applesoft Manual will tell you why. *Apple Mechanic* has two programs that make writing shape tables a snap. To learn more about shape tables, their usage and structure, read *How to Use Shape Tables* starting on page 27.

TWO SHAPE EDITORS

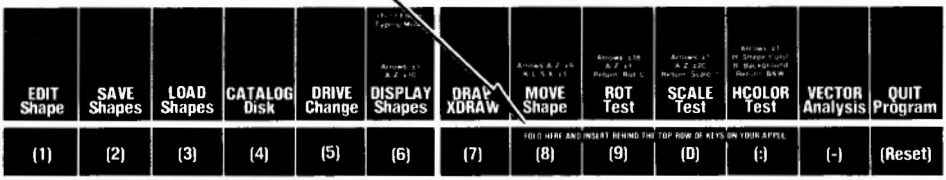
Two different shape-writing programs are on the *Apple Mechanic* disk. ***Shape Editor*** lets you create and store up to 12 shapes at a time, each one up to 48 x 63 plots in size. These shapes can be utilized by your own Applesoft programs. ***Font Editor*** lets you make hi-res characters or pictures that can be utilized by the *Xtyper* and *Hi-Writer* programs (both on this disk), as well as your own programs.

ERROR MESSAGES

Most error messages will relate to disk errors occuring when you are loading files. "FILE NOT FOUND" means the file you requested is not on the disk (or you spelled its name wrong) or you are using a disk that does not have the *Cursors* file on it. Other error messages will be referenced by number. See your Beagle Bros *Peeks & Pokes* Chart or Applesoft Manual for ONERR error message codes. If you suspect you have made an error, remove the ONERR statement from the beginning of the program you are working with. Then you can list the offensive line and get a clue regarding what's happening.

THE SHAPE TABLE KEY CHART

Use the Shape Editor side of the Key Chart that came with your *Apple Mechanic* disk. FOLD IT and insert it behind the top row of keys on



your Apple. The 1-6 keys apply to the *Shape Editor* and *Font Editor* programs. The 7 through hyphen keys are for analyzing individual shapes with the *Shape Analyzer* program. Several Key Chart options are common to all three programs:

(2) SAVE EXISTING SHAPES

(Note: SAVE is not available from the *Shape Analyzer* program.)

After you have created one or more shapes with *Shape Editor* or *Font Editor,* your shapes are in your Apple's memory, but not on disk. To save your shapes onto disk, press the **2** key. You will be asked for a title for the shapes you are saving. Any legal file name is o.k., but it helps to add a descriptive word like "FONT" or "SHAPES" to each name to keep them straight. You can always rename files later.

You will recognize shape tables or shape-fonts in your catalog mainly by the title you have given them. There will also be a "B" for Binary to the left of the sector number. You cannot BRUN these files, only BLOAD them and then use them from your own Applesoft programs or *Xtyper* or *Hi-Writer.*

If you find yourself accidentally in the Save Mode, hit the RETURN key (with no file name) to escape, *or* enter one of the Key Chart keys (and return).

(3) LOAD NEW SHAPES

Select **3** to load a new shape table for editing or analyzing. The *Shape Analyzer* program can handle *any* Apple shape table, but *Font Editor* and *Shape Editor* are very specialized. Thus, this warning:

LOAD ONLY SHAPE TABLES MADE WITH THE EDITING PROGRAM YOU ARE USING. OTHERS WILL EITHER NOT LOAD OR NOT BE EDITABLE.

(4) CATALOG THE DISK

Back to the Key Chart— A **4** keypress lets you catalog the disk drive most recently specified; handy if you want to inspect your file names. You will be temporarily switched to all-text mode for the catalog. Hitting any key will then return you to hi-res plus text.

(5) DRIVE/SLOT CHANGE

Seleting **5** lets you change the drive that will next be accessed by the program. Hitting RETURN defaults to the slot or drive value printed on the screen. Do not select a drive or slot that is not connected or the program will hang.

The SHAPE EDITOR Program

This program lets you draw hi-res shapes by plotting them on the screen from the keyboard. It will then temporarily store the drawings in your Apple's memory. When finished, you can save the drawings to disk in the form of a 12 shape (maximum) shape table for use in your Applesoft programs.

ONLY SHAPE TABLES MADE WITH THIS PROGRAM CAN BE EDITED WITH THIS PROGRAM. OTHERS *MAY* LOAD, BUT WILL NOT BE COMPATIBLE OR EDITABLE.

To begin, insert the *Apple Mechanic* disk in your drive, and type "RUN SHAPE EDITOR". When the program title is on the screen, six Key Chart keys control the Editor:

1: EDIT a Shape
2: SAVE Existing Shapes*
3: LOAD New Shapes*
4: CATALOG Disk*
5: DRIVE/SLOT Change*
6: DISPLAY Shapes

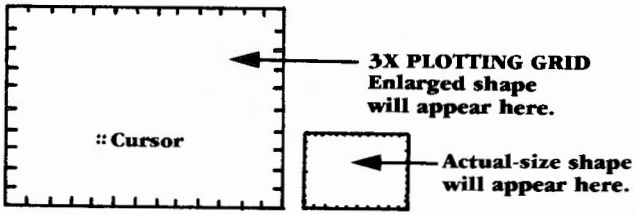
*Covered under *Key Chart*, page 23

You can usually "escape" to this part of the program by pressing the **ESC** key. When the program title is visible, you will see all 12 shapes currently in memory on the hi-res screen. If you are just starting, you will quite likely see the Editor's default shapes, 12 white dots. Each dot is actually a hi-res *shape* in a 12-shape shape table. Load a set of shapes, like *Demo Shapes*, as a test (press **3**, enter the name and wait while they load). You can *replace* each of the shapes you see with any drawing you want to create.

Due to the variety of shapes you might produce, the 12-shape menu display could get a bit messy. Unusually large shapes could overlap or "wrap around" and partially print on the opposite side of the screen. No problem, however; this display is for reference only. You can see each shape *alone* by selecting Key Chart Option **6** (more later) or by hitting **N** now to temporarily remove the text from the lower screen. Any key restores the text.

(1) EDIT A SHAPE

First, you need to choose which shape, 1-12, you are going to create. Actually you are going to *re-draw and replace* the shape you select. After pressing **1**, you will see a shape on the screen, perhaps a dot if you are just starting, both DRAWN and XDRAWN (these two often look the same), with its shape number below. To advance to the next or previous shape, use the Right or Left **Arrow** key. Shape #1 will follow #12 and vice versa. Another way to select a shape is to type the shape number you want, **1-9**. Shapes 10, 11 and 12 are selected for viewing/editing with the **Zero, Colon** and **Hyphen** keys. You can *animate* between any shapes you choose using these twelve top-row keys (for example, type 4-2-4-2-4-2-). With the shape or shape number that you want to redraw on the screen, press RETURN.



Now you see two rectangles on the screen and flashing cursor made of dots. Notice that you can move the cursor left & right with the **Arrow** keys and up & down with the **A** and **Z** keys. You will be creating a shape in a magnified version (3 times actual size) by moving this cursor inside the larger rectangle. Simultaneously, you will see the shape plotted in actual size in the smaller rectangle. Move the cursor, if necessary, to its starting point (allowing room for your drawing) and hit RETURN.

Now you have a *solid* flashing cursor (with a dot in the middle representing your drawing's start point). A *solid* flashing cursor means that the next vector (dot) will **PLOT AND MOVE** (up, down, left or right). A *dotted* flashing cursor means the next vector will **MOVE ONLY AND *NOT* PLOT.** To switch between plot and move, press the SPACE BAR. Notice the word "PLOT" on the screen changes to "MOVE" and back. Draw your shape with the A, Z and Arrow keys. Your drawing will consist of a number of "vectors" (visible or invisible lines one-plot long). All vectors, plotted or not, *must move.*

Shape Editor limits shapes to the size of the plotting grid, 48 x 63 hi-res plots, and to 999 separate vectors. You will encounter some appropriate flashing and/or beeping if you approach these limits. The 999 is determined by the variable MAX near the beginning of the program listing. Change it if you want.

(X) ERASE

To erase parts of shapes as they are being drawn, use **X** as a "back-space" key. You cannot erase *pre-plot* dots with this feature.

(P) PRE-PLOT

It is best to plot each shape as efficiently as possible* using the fewest number of vectors. If you want, you can *pre-plot* your drawing on the screen without regard to efficiency and then *trace over* the sketch for your final drawing. *After* you have selected the shape to be drawn and positioned the cursor, and *before* you hit RETURN, select **P**. Now you can draw on the screen with a series of dots. The actual-size drawing in the smaller rectangle will appear as it normally would. To erase a dot, you need to plot on top of it (you can leave extra dots on the screen; they will have no effect on your final drawing). When finished pre-plotting, hit RETURN. The actual size sketch will disappear but the 3X version will remain. Now you can trace the dots to create your shape.

*One more note on efficiency: Don't give it too much thought. An "inefficiently" drawn shape will take up some extra memory all right and take your Apple maybe a micro-second longer to draw. You can probably afford both.

(I) IMPRINT

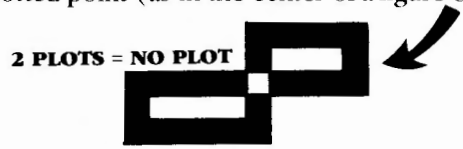
After you have selected the shape you want to edit, you may imprint any existing shape on the drawing screen by hitting **I**, selecting the shape to be imprinted (with the Arrow keys or 1-9, zero, colon or hyphen), and hitting RETURN. Imprinting is done if you want to create a shape similar to another (for animation, perhaps) by tracing over the imprint. The imprint is repositionable using the A, Z and Arrow keys. Repositioning is done to center the imprint in the rectangles or to align the imprint with your in-progress drawing.

To *keep* the imprinted image on the screen and start or continue plotting your shape, hit RETURN. To *erase* the imprint, hit ESC. The imprint will be erased from the small (actual size) rectangle, regardless of which option you select.

The imprint in the larger rectangle is XDRAWN with SCALE=3. You will notice that enlarged shapes seem to be a bit inaccurate. Ends of lines go too far or are bent at their ends, and dots appear as short lines. This is just the nature of scaled-up shapes. With a little practice, you'll get used to them and be able to use them as models for drawing. If the drawing you are creating is already in progress on the screen, you will see the imprint in *reverse* where it crosses your picture. In the actual-size drawing, the imprint and your drawing could actually cancel each other out temporarily.

PLOTTING TIPS

Lines that cross each other can be confusing at first. A plotting vector (solid cursor) will normally *plot*. But a vector that plots over an already-plotted point (as in the center of a figure-8) will *unplot* the



point. You may ignore this problem if you are going to mainly DRAW your shapes from your programs, but if you are going to XDRAW or animate them, the flaw will show. To move the plotting cursor *without* changing the drawing under it, use the SPACE BAR to select the dotted (non-plotting) cursor. And remember, you can always correct mistakes by backing up with the **X** key.

Positioning the cursor before plotting is usually necessary only on *large shapes* that might approach the limits of the drawing rectangle. If you are planning a shape that is going to go off to the right, for example, you should position the cursor near the *left* edge of the rectangle *before plotting*, giving you room to draw. The cursor starting position will remain the same until you change it.

The last point of a shape is where the *next* shape will start if your program uses a command like "XDRAW 1: XDRAW 2". In this example, Shape #2 will be drawn on the screen starting where Shape #1 stopped. If you will always be using plotting coordinates in your programs, as in "XDRAW 1 AT 10,10: XDRAW 2 AT 10,99", you don't need to be concerned where your shapes end.

You may use less than 12 shapes by simply ignoring the ones you don't want. The default dot shapes take up only two bytes apiece. **To create shape tables with more than 12 shapes, use the *Font Editor* program, which allows 95 shapes.**

(RETURN) FINISHING A SHAPE

Usually, you won't care where your shape *ends*. Just draw until it looks the way you want (especially in the actual-size version), and hit RETURN. When asked, "READY TO WRITE SHAPE INTO TABLE?", answer **Y**. If you had answered **N**, you would have been allowed to continue drawing. If you want to start over, answer **N** and then **ESC**.

Now your drawing will be "written" into the shape table. The more complex your shape, the more time this will take. The main title will reappear when writing is complete. Create as many shapes as you like, from 1 to 12. Just select Option **1**, and select the shape you want to redraw next.

(6) DISPLAY EACH SHAPE

Selecting Key Chart Option **6** when the main title is visible will let you view each shape individually, DRAWN and XDRAWN. Use the **Arrow** keys to view different shapes, and RETURN to see all twelve.

How to Use Shape Tables

This section covers the basics of using shape tables. The commands are simple and the results are amazing! List the *Shape Table Demo* to learn much much more. And above all, **experiment!** Get in there and make some mistakes (heck, I can't teach you *everything*).

LOADING YOUR SHAPE TABLE

First, set ROT & SCALE to their minimum values, 0 and 1. Then decide where in your Apple's memory to load your shape table. For now, let's pick location 25,000; it's a nice round number and out of the way of other goings-on. Set a variable in your program (for example, SH) equal to 25000 and load your shape table like so—

```
10  ROT= 0: SCALE= 1
20 SH = 25000:D$ =  CHR$ (4)
30  PRINT D$;"BLOAD SHAPES,A";SH
```

A shape table is a Binary file, so it must be BLOADed (but never BRUN). To play it safe, always tell the shape table where to load by specifying the location as part of the BLOAD command. In the example above, you could have used "**,A25000**" for the same results. While programming, you won't want to wait while your shapes load every time you run the program, so (after you have run it once) put a temporary "REM" in front of your BLOAD statement.

On page 45, there is a discussion regarding *More Room for Hi-Res Programs*. If you load your Applesoft program, as suggested there, at location 24576 ($6000), you will need to load your shape table elsewhere or one will erase the other. Location 2048 ($800), the normal *program* location, would be appropriate and allow you the most space. If you like round numbers, set SH equal to 3000.

SETTING THE SHAPE TABLE POINTER

Now you need to to tell your Apple where your shape table is (location 25,000) in memory. In your program, do two pokes—

```
40  POKE 232,SH -  INT (SH / 256) * 256
50  POKE 233, INT (SH / 256)
```

Assuming you have set SH equal to 25000, these two pokes put that number into the *Shape Table Pointer* at memory locations 232 & 233. If you run the above program now, that pointer will be set. If you *forget* to do the two pokes in your *next* shape table program, the program *might* work fine, because the pointer *stays set*, even surviving the often fatal "NEW" and "FP" commands. But tomorrow your Apple won't know where your shape table is stored. When you turn off your Apple or re-boot, the Shape Table pointer will "point" to the wrong place. The DRAW command will probably produce random scribblings, much as if you hadn't even loaded your shape table. So, *always* include the two pokes at the beginning of your shape table programs.

USING MORE THAN ONE SHAPE TABLE

To alternate between shape tables, simply re-poke a new location into the shape table pointer at 232-233 each time you change.

HCOLOR

To DRAW a shape, you need to specify a color by setting HCOLOR equal to a number, 0-7. If you don't, your shape might be drawn in HCOLOR 0 (black) and you won't see it. Here are Apple's hi-res colors.—

HCOLOR=0 or 4: Black
HCOLOR=1: Green
HCOLOR=2: Violet
HCOLOR=3 or 7: White
HCOLOR=5: Orange
HCOLOR=6: Blue

DRAW AND XDRAW

DRAW means just that, *draw* a shape from a shape table. Use the command like this—

```
99  DRAW 3 AT 100,90: REM (AT X,Y)
```

The first number after DRAW is the number, 1-255, of the shape to be drawn. The next two numbers are the horizontal and vertical hi-res coordinates of the starting point of the shape. If you get an ?ILLEGAL QUANTITY error, you have either attempted to draw outside of the screen's limits *or* you have specified a shape number larger than the number of shapes available.

The **XDRAW** command works like DRAW, but ignores HCOLOR and plots each dot of your shape in the *opposite* color of the dot that is being plotted over. XDRAW is handy for animation because you can make a shape travel over and be seen against multi-colored backgrounds.

BLACK (0/4) is opposite WHITE (3/7)
GREEN (1) is opposite VIOLET (2)
ORANGE (5) is opposite BLUE (6)

HI-RES COORDINATES

To tell a shape where to appear, specify the coordinates (horizontal, vertical) of the shape's starting point. Don't specify a start location outside of the screen limits, or your program will crash. Here are the hi-res limits for Apple hi-res—

Horizontal: 0 to 279
Vertical: 0 to 191
Vertical with 4 text lines: 0 to 159

DRAWING WITHOUT COORDINATES: After your first shape is drawn, you don't need to specify coordinates for the next DRAW or XDRAW, but you probably will want to. Each successive shape drawn without coordinates (as in "XDRAW 3: XDRAW 1: XDRAW 9") will begin *at the point where the previous shape stopped.* Eventually, using consecutive commands like this, shapes will usually "wrap around" and begin plotting on the opposite side of the screen.

SCALE

You can enlarge a shape by setting SCALE equal to a number, 2-255 or zero. A SCALE of zero is equivalent to a SCALE of 256. Scaled-up shapes are of limited use because the plotting vectors are *lengthened only* and spread apart as they are enlarged. They often end strangely-bent at right angles, depending on how the shape was created. In short, scaled-up shapes usually just don't look right.

ROT

You can rotate a shape by setting ROT before you draw—

ROT=0: Normal
ROT=16: Rotated 90° Clockwise
ROT=32: Rotated 180°(upside down)
ROT=48: Rotated 270°Clockwise

ROT values between the above only apply when SCALE is set larger than minimum (1). ROT values greater than 64 simply repeat the cycle until ROT=255.

MOVING A SHAPE

There is more than one way to move a shape. Basically you have to *draw* the shape, *erase* it, and *redraw* it in a new position. This XDRAW example moves a shape across the screen—

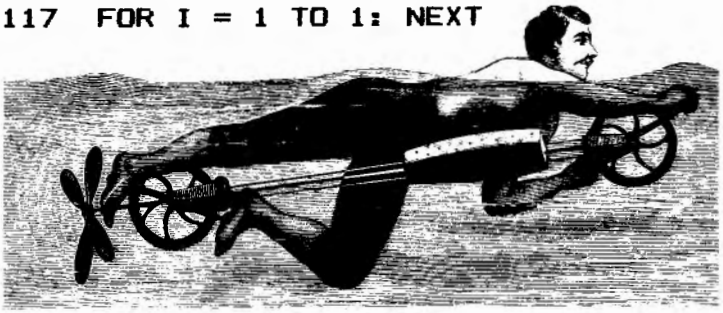
```
100  FOR X = 0 TO 279
110  XDRAW 5 AT X,100: REM DRAW SHAPE
115  XDRAW 5 AT X,100: REM ERASE SHAPE
120  NEXT X
```

With DRAW, you erase the shape by re-DRAWing it in the background color—

```
100  FOR X = 0 TO 279
110  HCOLOR= 3: DRAW 5 AT X,100: REM DRAW
115  HCOLOR= 0: DRAW 5 AT X,100: REM ERASE
120  NEXT X
```

Experiment with adding different delay loops between drawing and erasing. For example add these lines—

```
113  FOR I = 1 TO 50: NEXT
117  FOR I = 1 TO 1: NEXT
```



OTHER HI-RES MANIPULATIONS

HGR— Clears Page 1 to black & reveals it for drawing.
HGR2— Clears Page 2 to black & reveals it for drawing.
HCOLOR=X: HPLOT 0,0:
CALL 62454— Clears hi-res in HCOLOR X

The following "switches" do not clear the screen, but *reveal* whatever is currently on the page 1 or 2 text, hi-res or lo-res screens:
POKE 49232,0— View graphics screen* (hi- or lo-res)
POKE 49233,0— View text screen
POKE 49234,0— View full graphics screen (hi- or lo-res)
POKE 49235,0— View graphics plus 4 text lines* (VTAB 21-24)
POKE 49236,0— View Page 1* (hi-res, lo-res or text)
POKE 49237,0— View Page 2
POKE 49238,0— View lo-res
POKE 49239,0— View hi-res*
*Automatically set by HGR

POKE 230,32— Allows drawing on page 1
POKE 230,64— Allows drawing on page 2
POKE 230,96— Allows drawing on page 3 (not directly viewable)

The three pokes above disregard which page is currently being viewed. Poking other numbers into location 230 produce unpredictable, often disastrous results. Try POKE 230,48 and do some hplotting, for example. Be prepared to lose your program and re-boot if you try other numbers.

PRINT PEEK(228)— Prints HCOLOR code (see *Peeks & Pokes*)
PRINT PEEK(231)— Prints current SCALE value
PRINT PEEK(249)— Prints current ROT value
CALL 62923— Stores last plotted coordinates at locations 224-226
PEEK(226)— VERTICAL position during last CALL 62923
PEEK(224)+PEEK(225)*256— Horizontal during CALL 62923

The FONT EDITOR Program

This program is similar in concept to *Shape Editor,* but specialized to create shape table character sets or *fonts* of type. Each "shape-font" consists of 95 hi-res characters that can be printed onto the screen by adding Applesoft commands to the *HI-WRITER* program. Or you can type the characters directly onto a hi-res picture or blank screen by running the *XTYPER* program. Shape-fonts may be redesigned or used as-is without editing. Any character may be redrawn as you like (as pictures, symbols or whatever), as long as it fits within the maximum dimensions specified by the Editor.

ONLY SHAPE-FONTS MADE WITH THIS PROGRAM CAN BE EDITED WITH THIS PROGRAM. OTHERS *MAY* LOAD BUT WILL NOT BE COMPATIBLE OR EDITABLE.

SHAPE FONTS vs. OTHER FONTS:

In your encounters with other disks, you will undoubtedly come across other hi-res type fonts. *Apple Mechanic's* shape table type is unique, however, and incompatible with most other Apple type. That is, you can't use the same programs to display or edit shape-fonts as you do to display or edit other kinds of hi-res type.

ADVANTAGES: Shape-fonts have the distinct advantage of being *proportionally spaced.* An "I", for example, doesn't need to occupy the same screen space as a "W". Word-spacing and the distance between lines of type is adjustable. And type doesn't have to be any certain height; it can range between 3 and 16 dots high, exactly as you designed it, and be positioned *anywhere* on the hi-res screen without regard to vtabs or htabs. Shape-font type can be *XDRAWN* too, so that each character is automatically printed in the opposite of the background color. And shape-font characters are fast— *Font Editor* creates characters that will be drawn from upper-left to lower-right, ending adjacent to the left edge of the next character to be printed. Therefore coordinates do not need to be specified when displaying each character-shape.

DISADVANTAGES: Shape fonts take up more memory and disk space than other hi-res fonts you may have seen, usually around 18 sectors for large-height fonts and 7 for small-height fonts (full 95-character fonts). You will find fonts on Apple Computer's *DOS Tool Kit,* for example, that only occupy 5 sectors. The difference is in the flexibility and attractiveness of the typefaces and resulting screen presentations.

LARGE FONTS vs. SMALL FONTS

All shape-fonts are classified as Large or Small. Small fonts correspond approximately in height to Apple's normal text characters. All font names on the *Apple Mechanic* disk are preceded by a "]" (shift-M). All are large-height fonts, except those with the word "small" in their names. Another way to recognize font size is by catalog sector size— Full 96-character large fonts' tend to be 16-20 sectors, and small

fonts 6-10 sectors. Large-font characters may be up to 14 x 16 hi-res plots in size. Small characters are limited to 7 x 8. The difference is the alignment of characters when printed with *Xtyper* and *Hi-Writer*. You may, of course, create small characters in a large font, but not vice versa. If you know how shape tables are constructed, you might like to know that the second byte is set to 1 for small fonts and 2 for large fonts.

RUNNING THE PROGRAM

To begin, insert the same Key Chart you used for *Shape Editor*. Boot the disk, if necessary, and "RUN FONT EDITOR". When the program title is on the screen, six keys control the program:

1: EDIT a Character
2: SAVE Existing Shapes*
3: LOAD New Shapes*
4: CATALOG Disk*
5: DRIVE/SLOT Change*
6: DISPLAY (Type) Shapes

*Covered under *Key Chart*, page 23

You can usually escape to this part of the program by pressing the **ESC** key. See the *Shape Table Key Chart* notes on page 23 for details on options 2-5.

When you run the program, you will see the plotting grid and the 95 characters currently in memory. If you are just starting, you might see the Editor's default "characters", 95 white dots. Each dot is actually a hi-res *shape* in a 100-shape shape table. If you wanted a font with just a few characters (only numbers, for example), you could start with this dot "font" and add only your special characters, or you can use the *Font Splitter* program (page 42). More likely, you will want to load one of the complete fonts on the disk (using Key Chart Option 3) and change a few characters.

THE PLOTTING GRID

After you have loaded a font with Key Chart Option 3, the character set will be displayed. To the left you will see a rectangular plotting grid for drawing characters at 10 times actual size. The grid's size depends on whether a large (14 x 16) or small (7 x 8) font was most-recently loaded. The dotted lines on the grid have no function other than to serve as visual guides for lower case, descenders and so on. Their use is up to you.

(1) EDITING CHARACTERS

After you have loaded a font (Key Chart Option 3), select the character to be edited by pressing the **1** key. A striped flashing cursor will appear on one of the characters. To move the cursor to the character you want to edit or replace, use the **Arrows** and **A** & **Z** keys. Below the character set, you will see the ASCII value of the key that will eventually type the character (for example, KEY "A", ASCII 65). The SPACE character, ASCII 32, is the only one not accessible from the Editor. Hit RETURN when the cursor is on the character you want to edit.

Now you have three options:

(I) GRID IMPRINT: Prints the selected character as a 10x blow-up on the grid; good for making minor changes since most of the character will already be drawn for you.

(E) ERASE/REDRAW: Temporarily erases the character from the screen so you can completely redraw it.

(S) SHADOW IMPRINT: Prints a "shadow" of a character on the 10x grid as a reference. The shadow is for reference or tracing purposes only and does not affect the final shape.

Select **I, E** or **S,** and stand by. You will now see a round dot cursor, either outlined or solid, flashing in the upper-left of the grid. The **A, Z** and **Arrow** keys will move the cursor within the confines of the grid. Pressing the **Space Bar** changes the cursor from *Solid* to *Outlined* and back, representing *Plot* and *No-Plot*. Play around to see how it works. Notice that the actual-size character in the font to the right changes as you plot on the grid. You won't have the problem of dots unplotting when you plot over them (as in *Shape Editor*), because you are not actually creating the shape yet, only sketching it. The actual shape vectors will be created by the Editor, so there is no need to plan an "efficient" plotting path. Continue plotting and unplotting until the character looks the way you want it.

The shape you have plotted is only temporary; it isn't in the Apple's memory yet, only on the screen. To place it in memory (into a shape table), hit RETURN, and sit back. A cursor will scan the grid and "read" the shape. Now you are asked to "SELECT CHARACTER TO REPLACE". The character you just drew can now replace *any one* of the 95 characters on the screen. Usually you will want to replace the same character you first selected, so just hit RETURN to insert the new shape data into memory.

If you start with a dot (blank) font, and *if* you draw a character other than the first character (ASCII 33), and *if* you quit the program and re-run it, you will see your shape *repeated* from the first shape position up to the point where you inserted your new shape. Well, what you see is what you get. That is one of the quirks of working with the blank dot font, since the "font" actually consists of just *one dot* referenced by all 95 characters' pointers. To play it safe, when starting with the dot font, edit the *lowest* numbered character available first.

(6) DISPLAY (TYPE) SHAPES

Font Editor features a one-line character typer for testing the appearance of your characters adjacent to other characters. Select **6** from the main menu and type as you normally would. The **ESC** key toggles between upper & lower case. **RETURN** or typing near the right margin will return you to the main menu. Because of program space limitations, *the backspace key does not function* in this mode. There is no way to move the cursor without typing, or to save an image either; this mode is for test purposes only. To type and save hi-res images, use the *Xtyper* or *Hi-Writer* programs.

The XTYPER Program

This program may be used to type onto the Page One hi-res screen using up to three shape-fonts at a time. You may want, for instance, to label a diagram or graph that you have plotted with another program. After you have added type to your picture (or to a blank screen), you can save the entire image to disk. To start, type "RUN XTYPER" and be sure your disk has at least one large or small shape-font on it including every font you intend to use.

THE MAIN MENU

Running *Xtyper* or selecting **ctrl-R** while typing displays the main menu and lets you load fonts from the disk, clear the hi-res screen, load & save hi-res images, and quit the program. Exit the menu and begin typing by pressing RETURN.

(1, 2 & 3) LOADING FONTS

You may load up to three large or small shape-fonts into memory in any order you wish. Select **1**, **2** or **3** from the menu and enter the name of the font. The font will then be loaded into memory showing you its start location after the font name (for example, "BLOAD FONTNAME, A18880"). You don't need to be concerned with this number. You may replace any font any time you wish from the main menu. If you *re-load* Font 1 or Font 2, any higher-numbered font in memory will automatically be reloaded at a new location. Therefore all three of the fonts you are using need to be on the same disk. If you attempt to load Font 2 or Font 3 when there is a lower number unused, your font number will be changed to the lowest unused number.

(X) CLEAR HI-RES SCREEN

Always select **X** to clear the screen if you have no image loaded on hi-res Page One. Otherwise you could encounter a "snowy" background (an uncleared screen) on which to type. If this happens, type **ctrl-R, X, Y** and **RETURN** (just follow the screen prompts).

(L) LOAD HI-RES PICTURE

Select **L** to load a hi-res picture from your disk. This could be a chart, a graph, a picture of your dog, or any hi-res image to which you want to add hi-res type.

(S) SAVE EXISTING PICTURE

After you have typed on a cleared screen or on your picture, select **S** from the main menu to save the entire image on disk. If you don't want a previous picture to be erased, don't use the same name when re-saving it. You will recognize a picture file in your catalog with its name preceded by a "B" for Binary and (usually) an "034" for 34 sectors. It helps if you name picture files with the word "picture" or similar identification in the title. We have used an "@" before *Apple Mechanic* picture file names.

Warning: Be sure your shape-font files are *locked*. If you accidentally enter a *font* name when saving a hi-res image, the font will be

erased from the disk if the font is not locked! You may lock and unlock files by first selecting **C** to catalog.

Note: To see any hi-res picture on Page One when you are not running a program, type **HGR** (return) and **BLOAD PICTURE, A$2000** (return), where "PICTURE" is the name of the image you want to see.

(C) CATALOG DISK (and Lock/Unlock)

Select **C** to catalog from the main menu. You will need to do this to check the names of fonts or hi-res pictures. You will be given the option of locking and unlocking files after the catalog.

(Q) QUIT PROGRAM

Selecting **Q** from the main menu lets you quit *Xtyper*. When you quit, you will see the message "GOTO 2000 TO CONTINUE". If you were to type "RUN" instead of "GOTO 2000", you would need to reload your fonts into memory.

(RETURN) BEGIN TYPING

Hit RETURN to begin or continue typing on the hi-res screen. You must have loaded at least one shape font to type.

TYPING WITH XTYPER

After you have loaded at least one font from the main menu and hit RETURN, you will see a flashing rectangular cursor on the screen that corresponds to the approximate height of the current font and the current case, upper or lower. Now you can type on the screen or use one of the control commands below. You will get best results if you type in white (any size) or in color (large fonts only) on the BLACK background areas of the screen (see *Color Change* below).

The following commands appear on your Key Chart:

(RETURN) CARRIAGE RETURN

Hitting **RETURN** while typing puts the cursor at the left margin on the next line down. To change the vertical distance the cursor will move, see *Leading Notes* below. If you continue typing to the right margin, your words will "wrap around" without dropping to the next line. With *Xtyper* you must hit RETURN just as you would on a typewriter.

(ESC) UPPER & LOWER CASE

The **ESC** key will toggle between upper & lower case. The cursor height will change accordingly. Since shape-fonts can vary in character height, the height of the cursor might not exactly match.

(ctrl-P, ctrl-T & ctrl-Q) NON-KEY CHARACTERS

Three Apple characters have no corresponding keys. With *Xtyper*, **ctrl-P** types a left-square bracket* (the *right*-square bracket* is typed with shift-M as usual). **Ctrl-T** types the underscore*, and **ctrl-Q** types the backslash.*

*Or the character programmed for that key.

MOVING THE CURSOR:

Eight commands move the cursor on the screen without typing or erasing. You must hold the **CTRL** key down while you move the cursor or a letter will be typed. If you have a spare finger, you may also use the **REPT** key for long distance moves. The Left & Right **Arrows** (ctrl optional) move the cursor left and right approximately *one character*. Move your fingers three keys to the left and use **ctrl-K & ctrl-L** to move the cursor left and right *one plot*. **Ctrl-A & ctrl-Z** move the cursor up and down approximately *one type line*. And one key to the right, use **ctrl-S & ctrl-X** to move the cursor up and down *one plot*. Due to the varying sizes of shape-font characters, the cursor height will not always exactly match the height of the characters being typed. The *top of the caps cursor*, however, will usually align with the *top of the letter* about to be typed or erased.

ERASING:

Proportional type has many advantages. One *disadvantage* is the difficulty of making corrections. *Xtyper* gives you several ways to erase (and typing a *space* over a character is *not* one of them). First, *the top of the cursor must be aligned vertically* with the top of a capital letter on the line of type on which you wish to make an erasure. Got it? Now, here are your options for erasing:

(ctrl-B) BACKSPACE/ERASE: Place the left of your cursor at the right of the character you want erased. Now use **ctrl-B** as a backspace/erase key. A vertical black* line will be drawn at the left edge of the cursor, and the cursor will move one plot to the left. Use the REPT key if you want (press CTRL, B *and* REPT), and continue ctrl-B-ing until you are through erasing.

(ctrl-E) CHARACTER ERASE: This method works only with white type on a black background— Align your cursor vertically as described above, and position its left edge so it *touches* any part of the character to be erased. Now type **ctrl-E** and your Apple will erase the character (in black). *Stencil Font* characters will often only be partially erased with ctrl-E due to their split design.

(ctrl-W) WIPE OUT: Typing **ctrl-W** will erase an entire line of type in black* from the top of the caps cursor down.

OVERTYPING: Another way to erase a character is to place the cursor over it, aligning the top and left edges, change the typing color (ctrl-O) to the color of the background, and *type over* the characters to be erased using the same characters and the same font.

*See *Default Erase Color* next page.

(ctrl-F) FONT CHANGE:

Selecting **ctrl-F** while typing will allow you to select a new typing font. You may not, of course, select a font number that has not been loaded into memory.

(ctrl-O) COLOR CHANGE:

Selecting **ctrl-O** while typing will allow you to select a new typing color. Due to Apple's hardware configuration, typing in colors other than white or black might present some surprises— Characters with thin verticals could look distorted, especially on a non-color monitor. Also, small characters are unreadable in colors other than black or white. And colored type or lines on a color background will tend to "vibrate" and produce "stair-stepped" edges. Apple's hi-res hardware is really designed to support colors on a BLACK BACKGROUND. *Xtyper* will not let you choose alternate color #7 (white) or #4 (black).

(ctrl-G) GRID:

Selecting **ctrl-G** while typing will display a 4 x 4 grid on your drawing. Hitting ctrl-G again will erase the grid. The purpose of the grid is to display the edges of the hi-res screen and to give some visual reference for aligning and centering type.

Warning: Don't type over the grid if you are going to erase the grid. If you do, you will draw (XDRAW) lines through characters that have come into contact with it.

(ctrl-R) RETURN TO MAIN MENU:

Selecting **ctrl-R** while typing will display the main menu, so you can save or load a picture, load a new font, clear the screen, catalog the disk or quit the program.

LEADING NOTES:

Leading (pronounced *"ledding"*) is the vertical distance between lines of type. *Xtyper* normally puts two lines of leading between characters, large and small. If you type a lower case "g" at the left margin, hit RETURN, and type an upper case "E", you will see a vertical gap of two plots between the characters. To change this default value of 2 to any other value, change the first non-REM program line in *Xtyper*. The variables LD(1), LD(2) and LD(3) represent the leading for the three fonts in memory. If you want,for example, *four* plots of leading when you hit RETURN with Font 2, make LD(2)=4. Note: Apple's normal text uses a corresponding leading of *zero*, putting one vertical space between characters, however, because the standard text font has no descenders (parts of letters like *g, j, p, q* and *y* that go below the base line).

DEFAULT ERASE COLOR:

Just in case you are doing a lot of typing on non-black backgrounds, you may change the second non-REM program line in *Xtyper*. The variable BG is the erase color used by ctrl-B and ctrl-W.

The HI-WRITER Program

Hi-Writer is meant to be used as part of *your* Applesoft programs. Its function is to do "live" printing of shape-font characters on the hi-res screen. So, rather than typing directly in hi-res with *Xtyper* and saving each finished picture as a 34 sector binary file, you can instead add Applesoft instructions to *Hi-Writer*, after Line 500, that accomplish the same thing at no cost in disk space other than that occupied by your program.

The HI-WRITER DEMO

Uncle Louie always said, "A listing is worth a thousand words," so RUN and then LIST the *Hi-Writer Demo* program starting at Line 500 (type "LIST 500—") and notice what makes it work. You probably won't want to change the main subroutines (Lines 90-500), but will just be adding your own program *after* Line 500.

LOADING FONTS

To begin programming, load the *Hi-Writer* program so you can customize it. LIST Lines 50-53 (type "LIST—53"):

```
50 FLAG = 0: REM (0=LOAD,1=DON'T)
51 FT$(1) = "]BLOCK"
52 FT$(2) = "]SMALL STANDARD"
53 FT$(3) = "]APPLE"
```

The variable **FLAG** in Line 50 tells the Apple whether or not to load the three fonts from Lines 51-53 into memory. If you have run the program recently, the fonts don't need to be loaded again (waiting for them is a drag), so change Line 50 to **FLAG=1** for programming and testing purposes. In the *final version* of your program, you *will* want the fonts to load when the program is run. You may even want to *delete* Line 50, leaving FLAG permanently set to zero.

Hi-Writer lets you access up to three fonts at one time. To begin, let FLAG=0 and enter the names of your three fonts at Lines 51-53. If you want only one or two fonts to be used, enter a blank as the font name. For example, **FT$(3)=""**.

When your program is first run (with FLAG=0), the three fonts will be loaded and their locations will be stored in memory. Whenever you change one or more of the fonts in Lines 51, 52, or 53, be sure the value of FLAG is set back to zero in Line 50. Otherwise your new font(s) will *not* be loaded when the program is run the next time. Likewise, whenever you *save* your program, be sure Line 50 reads **FLAG=0** *before* you save it! Remember, you can always delete Line 50 and your fonts will be loaded every time the program is run.

> ***HI-WRITER* WILL NOT FUNCTION WITH FONTS NOT CREATED WITH THE *FONT EDITOR* PROGRAM. DON'T EVEN *THINK* ABOUT TRYING IT.**

PROGRAM VARIABLES

Hi-Writer has certain variables built into Lines 90-500. Copy, format, color and screen position may be changed by setting values for these variables and then doing a **GOSUB 100**.

These variables appear on your Key Chart for reference—

A$: Words to be printed
FT: Current Font Number (1-3)
FT3$: Name of new Font 3 to be loaded during program*
VT: VTAB (1-24)
HT: HTAB (1-40)
IN: INVERSE (1=Yes, 0=No)*
CT: Auto-Center (1=Yes, 0=No)*
HC: HCOLOR of Type (0-7)
CL: HCOLOR for Clearing Screen (0-7)
RT: ROT Value for Printing (0-3)*
* Reset to Zero after each GOSUB 100

A$: PRINTING WORDS

To print on the screen, set a string **A$** equal to the words you want printed and then GOSUB 100. For instance:

```
52Ø A$ = "HELLO": GOSUB 1ØØ
```

The above command will print "HELLO" somewhere on the screen in the font number (FT) and screen position (VT & HT) *most recently specified.*

UPPER & LOWER CASE

If you can enter lower case directly from the keyboard (you *don't* need lower case hardware), skip this section. If you can't, run out and buy a copy of the *Program Line Editor* disk. Direct upper & lower case keyboard entry is just one of its multitude of advantages.

Next best, you can implement "@" and **GOSUB 90**. Every time the routine at Line 90 encounters an "@", it will change from upper to lower case or vice versa with each *new line* automatically reverting to upper case. For example:

```
59Ø A$ = "BƏEAGLE ƏBƏROS": GOSUB 9Ø
```

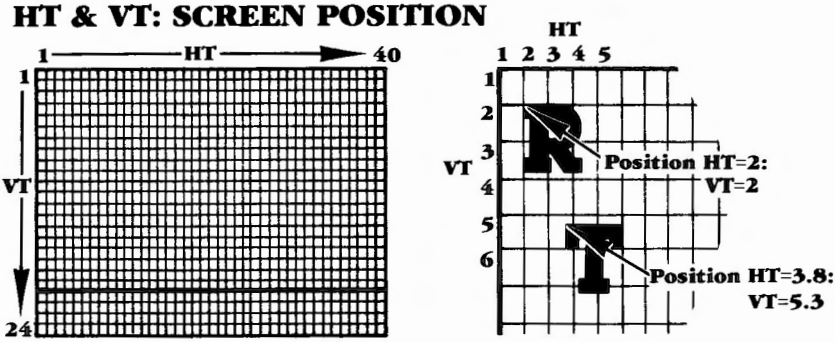
The above statement will cause **Beagle Bros** to be printed in upper & lower case. You can re-program "@" to be any character you want (in case you want to actually *print* some @'s on the screen) by changing Line 54's string **UL$**. For example, Line 55 could let **A$="/"**. Then **A$="S/MITH /B/ROS"** (GOSUB 90) would print as **Smith Bros**.

FT & FT3$: FONT SELECTION

Set the variable **FT** equal to **1, 2** or **3** any time you want to change fonts. Asthetically, three fonts in one graphic presentation is maybe one too many, but you may replace Font 3 as many times as you wish from within your program by entering **FT3$="FONT NAME": GOSUB 100.** There will be a pause while the new font is loaded

from disk. The only way to replace Fonts 1 & 2 is to change program Lines 51 & 52, reset FLAG in Line 50 to zero and re-run the program.

HT & VT: SCREEN POSITION



Each A$ word-string will be written on the screen with its *upper-left* corner at the htab/vtab position designated by the variables **HT** and **VT**. Decimal fractions are allowed.

HT (1-40) determines horizontal position. **VT** (1-24) determines vertical position. If new HT and/or VT values are not specified, new words will continue where previous ones stopped. Words printing off the screen will wrap around to the other side. Large-font characters are about 2-VT×2-HT in size. Small characters about 1×1.

Due to proportional spacing, it is difficult to predict the length of a word, so a little trial and error might be in order to produce attractive screen layouts. Enter a test program line, see how it looks by running the program, and make your changes appropriately. If you put a temporary "GOTO xxx" at Line 510 (where xxx is the line number you are working on), you won't have to wait through early parts of your program that are completed.

CT & GOTO 400: CENTERING

There are two ways to center type left & right on the screen with *Hi-Writer*— **METHOD 1** is to set variable CT equal to 1 and the program will automatically center that line of copy. For example:

```
550 CT=1:VT=8.2:A$="FRIED EGGS":GOSUB 100
```

The latest HT value will be cancelled and there will be a half-second or so pause before the centered line is printed. If this pause is objectionable, use **METHOD 2:** There is an HTAB Centering Calculator built into the program at Line 400 to be used only *while programming* (this routine was deleted from the *Demo* program). You must have run the main program at least once (the fonts must be loaded into memory) for the HTAB calculator to work. To use it, specify the string A$ that you want centered and its font. Then type "GOTO 400"

```
A$="PITTSBURGH": FT=3: GOTO 400
```

This is a *direct keyboard command* (no line number) that will return a value for HT of, perhaps, 14. This tells you to let HT=14 in your program to center the copy "PITTSBURGH" typed in Font 3.

This routine will not work with lower case copy if you are unable to enter lower case directly from the keyboard. You might want to delete this routine, Lines 400-499, to save memory (see *Out of Memory?* below).

IN: INVERSE

Setting variable **IN** to **1** means the words that follow will be printed in inverse. IN is automatically reset to zero (meaning normal type) after each GOSUB 100, and will not function on rotated type. It is best to add at least one space both *before* and *after* a word to be inversed, giving it a framed appearance.

HC: HCOLOR

The variable **HC** determines the HCOLOR of the words that follow. See the *Color Change* notes in the *Xtyper* instructions regarding conflict of interest between adjacent Apple colors.

RT: ROTATION

The variable **RT** will print your words rotated, thanks to Apple's ROT function. A rotation of **1** will rotate type 90° clockwise. **3** will rotate it 90° counter-clockwise. And RT=**2** will produce upside down type (just what you've always wanted!). Rotation can be tricky, so plan ahead. Remember, HT and VT determine the *upper-left start point* of the first letter printed. Rotated type cannot be inversed.

CL: CLEARING THE SCREEN

Specifying a value 0-7 for the variable **CL** followed by a GOSUB 100 will clear the screen in the corresponding HCOLOR. VT and HT will automatically be reset to 1.

COMBINING TEXT & HI-RES

Apple's good old text screen is always available for you to use with its HOME, PRINT, HTAB, VTAB, etc., commands—

POKE TXT,0 in your program reveals the lower sixth of the text screen, VTAB 21-24, leaving the upper screen for your hi-res *Hi-Writer* commands.

POKE FULL,0 will switch you back to full hi-res with the text screen "hidden" behind.

TEXT will switch you entirely over to a text display without erasing your hi-res image.

POKE HIRES,0 will switch you back to hi-res without clearing the screen (HGR would do that if you wanted). See the *Hi-Writer* demo for examples.

OUT OF MEMORY?

Your program, including *Hi-Writer*, must not be so large that it begins overwriting the hi-res screen. If it does, you will get uncalled-for little horizontal stripes on your hi-res image, really making a mess of things. If you want, just insert—

```
5000  PRINT  CHR$ (4);"RUN NEXT PROGRAM"
```

in your program and an entire *new* version of *Hi-Writer* will continue.

To save considerable program space, delete Lines 400-499 (center-calculator) and Lines 90-99 (lower case converter).

NON-KEY CHARACTERS

CHR$(95) is the underscore,* CHR$(92) is the backslash* and CHR$(91) is the left square bracket.* Remember, you can join strings, as in A$="ABC"+CHR$(91).

*or the font character programmed for that key

ERRORS

There is an ONERR GOTO 450 in Line 55 that prints a code when an error is encountered (see your *Peeks & Pokes* Chart). Most errors, other than ?Syntax Errors will occur in line numbers in the 100's, but the *cause* of the error will be elsewhere; you probably tried to print beyond the limits of the screen or with an illegal color. Take note of **the last hi-res character printed** on the screen; it was just after that that your error occured. Type "TEXT" or hit RESET to find and correct the error.

FONT SPLITTER

Run *Font Splitter* to reduce the number of characters in a font. You may, for example, want to save disk or program space by removing all of the lower case characters from a font. Follow the screen prompts and let your Apple do the rest of the work.

PAGE COPY

To clear the screen and load a hi-res picture onto PAGE 1:

10 HGR: PRINT CHR$(4); "BLOAD PICTURE, A$2000"

To clear the screen and load a hi-res picture onto PAGE 2:

10 HGR2: PRINT CHR$(4); "BLOAD PICTURE, A$4000"

These commands may be used in a program as shown above, or typed directly (with no line number). For example:

HGR (return) **BLOAD PICTURE, A$2000** (return)

With the *Page Copy* text files, you can *move* images already in memory from one page to another with these commands:

EXEC PAGE COPY 1=2 (return) moves Page 2's image to Page 1.

EXEC PAGE COPY 2=1 (return) moves Page 1's image to Page 2.

Either of these EXEC commands will work in a program, but *only* if it is the *last command* in that program. To move pictures from within a program, use Apple's CALL -468 after poking locations 60-67:

```
20  INPUT "MOVE IMAGE FROM PAGE:";A
30  INPUT "  MOVE IMAGE TO PAGE:";B
40  POKE 60,0: POKE 61,A * 32
50  POKE 62,0: POKE 63,A * 32 + 32
60  POKE 64,0: POKE 65,B * 32 + 32
70  POKE 66,0: POKE 67,B * 32
80  CALL  - 468
```

Note: CALL-468 will often leave blemishes on the "from" page, especially if you are working with invisible "Page 3" ($6000-$7FFF).

SHAPE ANALYZER

This program lets you load a shape table and analyze its individual shapes. You can see each shape with selected values for ROT, SCALE, shape-HCOLOR and background-HCOLOR. You can also DRAW, XDRAW and move shapes and see a text screen analysis of each shape's vectors.

RUNNING THE PROGRAM

Insert the *Apple Mechanic* disk and type "RUN SHAPE ANALYZER" to start the program. You will be asked for the name of a shape table to be loaded. Any shape table qualifies, including shape-fonts, and shape tables made without using *Shape Editor* and *Font Editor*. *Non*-shape table binary files will also load, but their analysis will, of course, be meaningless. After the shape table loads, you will see Shape #1 on the hi-res screen with its description below. (Note: A shape-font's Shape #1 is a *space*, and won't show.)

You can usually return to this part of the program by hitting **ESC** or RETURN. Using the Shape Editor side of your Key Chart, the following top row keys control *Shape Analyzer*:

3: LOAD New Shapes*
4: CATALOG Disk*
5: DRIVE/SLOT Change*
6: DISPLAY Each Shape
7: DRAW/XDRAW
8: MOVE Shape
9: ROT Test
0: SCALE Test
Colon: HCOLOR Test
Hyphen: VECTOR Analysis

*Covered under *Key Chart*, page 23

ANALYSIS MODES

The **8** through **Hyphen** keys enter you into a temporary "mode" where you may adjust the selected characteristic of the shape being displayed. The **ESC** key will exit you from that mode. **RETURN** will set the default value where appropriate.

(6) DISPLAY SHAPES

Pressing the **6** key or an Arrow key will let you select a new shape for hi-res display. The left & right **Arrow** keys decrease & increase the shape number by *one*. The **A & Z** keys decrease and increase the shape number by *ten*. The shape's number will appear below the shape.

(7) DRAW/XDRAW

The **7** key will change the current display from DRAW to XDRAW and vice versa. The current status will be displayed in inverse on the top line of text below the shape.

(8) MOVE SHAPE

Selecting **8** will let you move the current shape on the screen. You

might want to do this if the shape drawn off the edge of the screen or behind the text at the bottom. The **A, Z** and **Arrow** keys move the shape *9 units* vertically and horizontally. The parallel **S, X, K** and **L** keys do the same *1 unit* at a time. An odd move-number (9) is used in order to analyze shapes in HCOLORS 1, 2, 5 and 6. Each of these colors shows only in an odd OR even column, so a color shape will appear differently depending its horizontal position.

(9) ROT TEST

Selecting **9** lets you change the ROT value for the shape being displayed. The **Arrow** keys increase and decrease ROT by *16* (90°). **A & Z** do the same in units of *1*. Hitting RETURN changes ROT to zero (normal). Note: Small changes in ROT do not necessarily have effect on shapes of small SCALE. See *Using Shape Tables* and your Applesoft Manual.

(0) SCALE TEST

Selecting **Zero** lets you change the value for SCALE with the **Arrow** keys. For an increase or decrease of 20, use the **A & Z** keys. Hitting RETURN resets SCALE to 1 (normal). Note: A SCALE of zero is equivalent to the *largest* possible SCALE, 256.

(:) HCOLOR TEST (HYPHEN Key on Apple IIe)

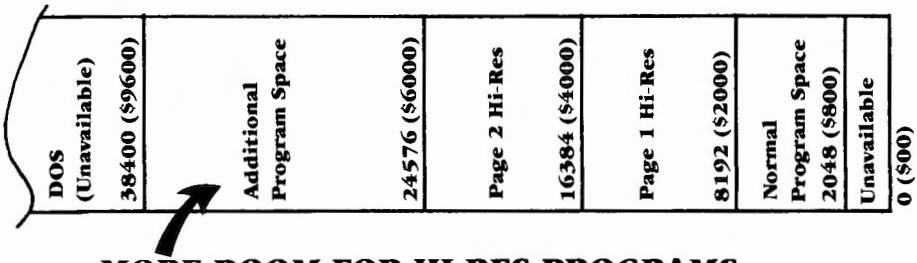
Hitting the **Colon** key will let you change the HCOLOR of the current shape. Since HCOLOR does not apply to the XDRAW command, you will automatically be placed in DRAW mode. The **Arrow** keys increase and decrease the HCOLOR value (0-7). If you select **B** while in this mode, you may similarly change the background-HCOLOR. **H** again lets you change the shape-HCOLOR. RETURN will reset the HCOLOR for the shape to 3 and the background to 0.

(-) VECTOR ANALYSIS (EQUAL Key on Apple IIe)

Hitting the **Hyphen** key will let you analyze a shape vector-by-vector. You will first see the shape table's memory location (16384 or $4000 for this program) and the values for Byte #1 and Byte #2. Byte #1 is the number of shapes permitted by the table. Byte #2 is normally wasted, but shapes made with our editing programs use it—1 symbolizes a small shape-font; 2 is a large shape-font; and 0 means a non-font. Enter a shape number (or hit RETURN for the current shape) and watch the screen. Reading left to right, you will see each vector of the shape, indicated by its value (0-255), and an interpretation of the moves within that vector. **Inverse** type indicates Move & *Plot*. **Normal** type indicates Move & *Don't Plot*. After the analysis, hit ESC to see a graphic display of that shape, or RETURN for another vector analysis.

Run the "APPLE MECHANIC" program*
and select Option "K" for
recent update information and
special notes regarding the Apple IIe.

* This program runs when you boot the Apple Mechanic disk.



MORE ROOM FOR HI-RES PROGRAMS

Why did they put hi-res in the *middle* of everything? Actually, I don't *care* why. I just want to know how to solve the problem it causes. You see, programs and pictures are stored in consecutive memory locations. An Applesoft program normally starts at location 2049 ($801) and can be nice and long, ending somewhere up in the 36,000 range, making it potentially over 30,000 bytes in size, a nice big program by my standards. HOWEVER, right in the middle of all this, at locations 8192-24575 ($2000-5FFF), are hi-res pages one and two. Anything drawn on these pages will zap any program long enough to even *think* of occupying part of that space. For example, an HGR can actually delete a large chunk of an Applesoft program. So really, a normally loaded Applesoft program must fit between 2049 and 8000 or so, depending on the number of variables involved in the program.

Solution #1: Use hi-res PAGE TWO, a nice friendly page. Just start off with an HGR2; then subsequent hplot commands and the like will be executed on page two. Now your program can be 14,000 bytes long instead of only 6,000 (approximate numbers). You can't use the four text lines below hi-res, however; your Applesoft program occupies that space (page two of text).

Solution #2: Set LOMEM just above page two with a "LOMEM: 24576" command at the start of your program. This will make Applesoft store variable information above page two instead of at the normal location between your program and page one. Use "LOMEM: 16384" if you don't need hi-res page two.

Solution #3: Load your entire program above both hi-res pages at location 24576 (or at 16384 if you are not going to use page two). Now you've got a *lot* more program space. To load at the new location, you need to poke a zero at the new location and new numbers into the start-of-program pointer at 103 and 104. If you insert the following program line at the start of your program, it will do the trick for you.

```
1 LOC = 24576 + 1: IF  PEEK (103) +  PEEK
     (104) * 256 <  > LOC THEN  POKE LOC -
     1,0: POKE 103,LOC -  INT (LOC / 256) *
     256: POKE 104, INT (LOC / 256): PRINT
      CHR$ (4);"RUN THIS PROGRAM"
```

Note: Be sure to SAVE your program with the above line added *before* you run it.

Apple Shape Table Vectors

$00/000: (end)	$20/032: U	$40/064: UUR	$60/096: UUR
$01/001: R	$21/033: RU	$41/065: RUR	$61/097: RUR
$02/002: D	$22/034: DU	$42/066: DUR	$62/098: DUR
$03/003: L	$23/035: LU	$43/067: LUR	$63/099: LUR
$04/004: U	$24/036: UU	$44/068: UUR	$64/100: UUR
$05/005: R	$25/037: RU	$45/069: RUR	$65/101: RUR
$06/006: D	$26/038: DU	$46/070: DUR	$66/102: DUR
$07/007: L	$27/039: LU	$47/071: LUR	$67/103: LUR
$08/008: UR	$28/040: UR	$48/072: URR	$68/104: URR
$09/009: RR	$29/041: RR	$49/073: RRR	$69/105: RRR
$0A/010: DR	$2A/042: DR	$4A/074: DRR	$6A/106: DRR
$0B/011: LR	$2B/043: LR	$4B/075: LRR	$6B/107: LRR
$0C/012: UR	$2C/044: UR	$4C/076: URR	$6C/108: URR
$0D/013: RR	$2D/045: RR	$4D/077: RRR	$6D/109: RRR
$0E/014: DR	$2E/046: DR	$4E/078: DRR	$6E/110: DRR
$0F/015: LR	$2F/047: LR	$4F/079: LRR	$6F/111: LRR
$10/016: UD	$30/048: UD	$50/080: UDR	$70/112: UDR
$11/017: RD	$31/049: RD	$51/081: RDR	$71/113: RDR
$12/018: DD	$32/050: DD	$52/082: DDR	$72/114: DDR
$13/019: LD	$33/051: LD	$53/083: LDR	$73/115: LDR
$14/020: UD	$34/052: UD	$54/084: UDR	$74/116: UDR
$15/021: RD	$35/053: RD	$55/085: RDR	$75/117: RDR
$16/022: DD	$36/054: DD	$56/086: DDR	$76/118: DDR
$17/023: LD	$37/055: LD	$57/087: LDR	$77/119: LDR
$18/024: UL	$38/056: UL	$58/088: ULR	$78/120: ULR
$19/025: RL	$39/057: RL	$59/089: RLR	$79/121: RLR
$1A/026: DL	$3A/058: DL	$5A/090: DLR	$7A/122: DLR
$1B/027: LL	$3B/059: LL	$5B/091: LLR	$7B/123: LLR
$1C/028: UL	$3C/060: UL	$5C/092: ULR	$7C/124: ULR
$1D/029: RL	$3D/061: RL	$5D/093: RLR	$7D/125: RLR
$1E/030: DL	$3E/062: DL	$5E/094: DLR	$7E/126: DLR
$1F/031: LL	$3F/063: LL	$5F/095: LLR	$7F/127: LLR

Shape Table Structure

Hello. This is extra information for advanced programmers, and is in NO WAY necessary in order to create shape tables with *Apple Mechanic* programs.

Each shape in a shape table is made up of a series of values, 0-255 ($00-$FF). Each value or *byte* can represent from one to three moves. Each move in the byte can either plot or not plot. The chart above shows the 256 possibilities. Reverse type signifies Move & PLOT. Normal type means Move & DON'T PLOT. Vector #0 ($00) means "Move Up" *only* if it is the *first vector* in a shape. Otherwise, Zero means END OF SHAPE. Notice how no vector can end in a Move-Up. Also, if a vector makes three moves, the third move cannot plot. To move up several times without plotting, you must use a vector like #193 ($C1)—Move-Right, Move-Up, Move-Left.

To construct a shape "by hand", draw it, dot by dot, on grid paper, then draw a line through each dot. Now find equivalent vectors from the table. Except for not being able to move up, you *could* construct any shape

URDL : PLOT & MOVE Up, Right, Down, Left
URDL : NO-PLOT & MOVE Up, Right, Down, Left

$80/128: UUD	$A0/160: UUD	$C0/192: UUL	$E0/224: UUL
$81/129: RUD	$A1/161: RUD	$C1/193: RUL	$E1/225: RUL
$82/130: DUD	$A2/162: DUD	$C2/194: DUL	$E2/226: DUL
$83/131: LUD	$A3/163: LUD	$C3/195: LUL	$E3/227: LUL
$84/132: UUD	$A4/164: UUD	$C4/196: UUL	$E4/228: UUL
$85/133: RUD	$A5/165: RUD	$C5/197: RUL	$E5/229: RUL
$86/134: DUD	$A6/166: DUD	$C6/198: DUL	$E6/230: DUL
$87/135: LUD	$A7/167: LUD	$C7/199: LUL	$E7/231: LUL
$88/136: URD	$A8/168: URD	$C8/200: URL	$E8/232: URL
$89/137: RRD	$A9/169: RRD	$C9/201: RRL	$E9/233: RRL
$8A/138: DRD	$AA/170: RRD	$CA/202: DRL	$EA/234: DRL
$8B/139: LRD	$AB/171: LRD	$CB/203: LRL	$EB/235: LRL
$8C/140: URD	$AC/172: URD	$CC/204: URL	$EC/236: URL
$8D/141: RRD	$AD/173: RRD	$CD/205: RRL	$ED/237: RRL
$8E/142: DRD	$AE/174: DRD	$CE/206: DRL	$EE/238: DRL
$8F/143: LRD	$AF/175: LRD	$CF/207: LRL	$EF/239: LRL
$90/144: UDD	$B0/176: UDD	$D0/208: UDL	$F0/240: UDL
$91/145: RDD	$B1/177: RDD	$D1/209: RDL	$F1/241: RDL
$92/146: DDD	$B2/178: DDD	$D2/210: DDL	$F2/242: DDL
$93/147: LDD	$B3/179: LDD	$D3/211: LDL	$F3/243: LDL
$94/148: UDD	$B4/180: UDD	$D4/212: UDL	$F4/244: UDL
$95/149: RDD	$B5/181: RDD	$D5/213: RDL	$F5/245: RDL
$96/150: DDD	$B6/182: DDD	$D6/214: DDL	$F6/246: DDL
$97/151: LDD	$B7/183: LDD	$D7/215: LDL	$F7/247: LDL
$98/152: ULD	$B8/184: ULD	$D8/216: ULL	$F8/248: ULL
$99/153: RLD	$B9/185: RLD	$D9/217: RLL	$F9/249: RLL
$9A/154: DLD	$BA/186: DLD	$DA/218: DLL	$FA/250: DLL
$9B/155: LLD	$BB/187: LLD	$DB/219: LLL	$FB/251: LLL
$9C/156: ULD	$BC/188: ULD	$DC/220: ULL	$FC/252: ULL
$9D/157: RLD	$BD/189: RLD	$DD/221: RLL	$FD/253: RLL
$9E/158: DLD	$BE/190: DLD	$DE/222: DLL	$FE/254: DLL
$9F/159: LLD	$BF/191: LLD	$DF/223: LLL	$FF/255: LLL

with vectors 0-7. In the interest of efficiency, however, include as many moves as possible in each vector. Here are three sample shapes:

SHAPE 1 **SHAPE 2** **SHAPE 3**

Let's work in hex and break each shape down into a series of vectors:

SHAPE 1: $05 (R), $00 (end)
SHAPE 2: $2D (RR), $36 (DD), $3F (LL), $24 (UU), $00 (end)
SHAPE 3: $0C (UR), $0C (UR), $15 (RD), $15 (RD), $06 (D), $00 (end)

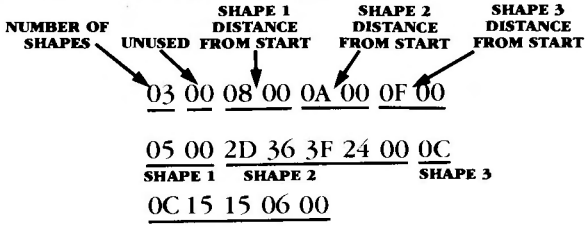
Those are the shapes. Now for the shape *table*— A shape table's first byte tells the computer how many shapes are in the table. The second byte is unused, so we'll make it zero. The next *pair* of bytes* says how far the first byte of the *first shape* is from the *start* of the shape table. The next pair does the same for the *second shape,* and so on.

*In hex, the second byte is most significant— For example, "08 00" means 8; "08 01" would be 264 (8 + 256×1); "0A 0F" is 3850 (10 + 256×15).

So here is our shape table at 24576 ($6000)—

```
6000— 03 00 08 00 0A 00 0F 00
6008— 05 00 2D 36 3F 24 00 0C
6010— 0C 15 15 06 00 00 00 00
```

And here's what it all means—



To type the above table, enter the monitor by typing "CALL -151" (return) and then "6000: 03 00 08 00 0A 00 0F 00 05 00 2D 36 3F 24 00 0C 0C 15 15 06 00" (return). Remember, these are all *hex* numbers (without $'s). To list the shape (from the monitor), type "6000.6014" (return). Exit the monitor by typing ctrl-C (return). Now your shape table is in memory at location 24576 ($6000). To save it, type "BSAVE SHAPES, A24576, L21" or "BSAVE SHAPES, A$6000, L$15". To load it, type "BLOAD SHAPES". See *Using Shape Tables* for more details.

Shape Table Program Notes

Here are some technical details regarding the fonts, cursors and locations used by the shape programs on *Apple Mechanic.*

Shape #1, the space character in each font, simply moves down and to the right without plotting and cannot be reprogrammed from *Font Editor.*

Shape #99 in the fonts move without plotting from the *bottom right* of a character to the *top left* of the next character. That's why you will see an "XDRAW 99" after each character is drawn.

Shape #100 is a "do nothing" shape, vector 25, handy for putting the hi-res cursor where you want it without moving or plotting.

CURSORS: This shape table is put to extensive use in *Apple Mechanic.* Shapes 1 & 2 are the solid and outlined plotting cursors in the editors. Shape 3 is a dot, used as a scanner for reading shapes and blown up to draw the grid in *Xtyper.* Shapes 4 & 5 are the upper and lower case cursors for *Xtyper.* Shape 6 is the shadow marker for *Font Editor.* Shape 7 is an underscore cursor that didn't get used. Shapes 5-7 all start *and end* at the upper left, making them good "flashers"; coordinates don't have to be specified every time they are drawn. Shapes 8 and 9 are *Shape Editor* cursors.

SH & SF are used as shape table locations. *Cursors* is loaded at SH ($4000 or $6000). The shapes being created or used are at SF ($4100 or $6100). To access one shape table or the other, the pointer at 232-233 is changed.

BYTE ZAP Instructions

Welcome to the world of tracks, sectors and bytes. *Byte Zap* is a program that gets you into the nitty-gritty of data storage, letting you examine and change one byte at a time. We will cover some of the more everyday uses of this program, how it works and how to use it as a tool. If you would like to get into disk repair and copy protection, please do yourself a favor and buy a copy of *Beneath Apple DOS* at your local computer store. This book says it all; half of it is still way over my head (and *I* wrote *Byte Zap!*), but I am always referring to it and learning from it.

$OME NOTES ABOUT HEX

Hex is a big pain; don't let anyone tell you differently. It is, however, a *beautiful* number system to use if you happen to be a computer. Most utilities that work with tracks and sectors use only hex. *Byte Zap* gives you the option of using hex *or* decimal. Hex numbers in the program are preceded by a dollar-sign and followed by the decimal equivalent in parentheses.

The hexidecimal system is constructed on a *sixteen* base instead of a ten base (like the decimal system). If we had eight fingers on each hand it would be easier (but that would *really* be a pain). Just remember, in hex you count 0-1-2-3-4-5-6-7-8-9-A-B-C-D-E-F-10-11-12, etc. (that's decimal 0-18). There's a little hex converter program on your *Peeks & Pokes* Chart that will convert numbers from 0 to 255 ($00 to $FF). All of the numbers we will use with *Byte Zap* will be in that range. In fact all numbers your *Apple* uses are in that range!

HOW APPLE DATA IS STORED

Let's start big and work our way down. A standard Apple floppy diskette is divided into 35 concentric *tracks* numbered 0-34 ($00-$22). Each track is further divided into 16 *sectors* numbered 0-15 ($00-$0F). Each sector is then divided into 256 *bytes* numbered 0-255 ($00-$FF). Bytes are made up of *bits* too, but this is where we get off.

A BYTE is a piece of magnetic data; I don't really understand; just think of each byte as a *number* between 0 and 255. This number can represent just that, a *number* (or part of a number), or an ASCII *character* (letters, numbers, etc.), or a BASIC *token* (command words like PRINT, GOSUB, etc.) or some kind of *machine-language* command. These bytes are *written* or re-written onto a disk when you *save* a program, *rename* a file, or perform any of the *writing* functions of DOS (Apple's Disk Operating System). When you *load* a program, you are only *reading* bytes off of the disk and into memory, not changing anything on the disk.

Byte Zap lets you change any byte anywhere on a disk. Therefore, you can do some "illegal" things, things you can't do with normal commands like LOAD, SAVE and RENAME. You can make completely flipped-out file names that backspace over themselves with inverse characters and so on. You can restore deleted files too

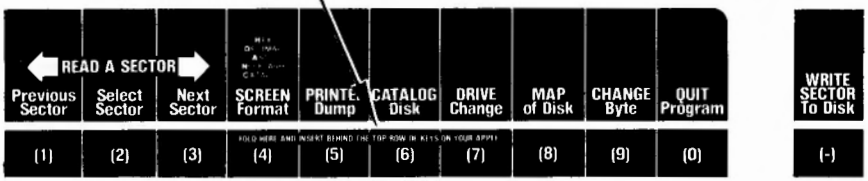
(check out that disk that was given to you; what files are on it that don't show in the catalog?). You can change DOS's commands and error messages, rename your greeting programs, restructure your catalogs, and (if you're not careful) **really make a big mess!** So make yourself a couple of test backup disks, and let's have some fun.

RUNNING BYTE ZAP

Run the program just like any other. After a copyright notice and a few seconds, you will be asked to insert the disk you want to be read. This can be any *normal DOS 3.3* disk. *Byte Zap* will not work on non-standard disks. After you hit RETURN, the program will read Track 17, Sector 15 (the first catalog sector), store it in your Apple's memory and display it on the screen.

THE *BYTE ZAP* KEY CHART

Use the *Byte Zap* side of the Key Chart that came with your *Apple Mechanic* disk. FOLD IT LENGTHWISE and insert it behind the top



row of number keys on your Apple. Functions and choices that are not on the chart will appear on the screen.

THE CURSOR

When a sector is displayed, you will see a flickering cursor on the screen. This cursor is on top of one of the 255 bytes currently being displayed. The width of the cursor (1, 2 or 3 characters) depends on the screen format, but the cursor always covers *just one byte.* The status display at the bottom of the screen tells you the byte number (where the cursor is), the value of that byte, and the ASCII equivalent (character) of the value.

To move the cursor, use the A, Z, Arrow and RETURN keys. The **Arrow** keys will move the cursor numerically up or down *one byte.* The flickering arrows that make up the cursor will point in the direction last moved. The **A** and **Z** keys will move the cursor up or down *one row* (a different number of bytes, depending on the screen format). **RETURN** will move the cursor numerically up or down *16 bytes* in the direction the cursor's arrows are pointing.

(1), (2) and (3) READ SECTORS

The **1** and **3** keys tell the program to read the *previous* and *next* sector on the disk in the active drive. Remember, sectors are numbered from 0 to 15 ($0-$F). After Sector 15 comes Sector 0 of the next track. As soon as the sector is read by your drive, the sector's 255 bytes will be displayed on the screen in the current format. On the left of the screen is a column of hex or decimal numbers representing the Byte Number of the number immediately to the right.

The **2** key will let you name the sector you want to read. You

may enter track and sector numbers in decimal or hex. Hex numbers must be typed following a dollar-sign. Hit RETURN after the number is typed. If you decide not to enter a number when asked, hit RETURN only. If you enter the wrong track number, hit RETURN only when asked for a sector. If you read the wrong sector, no big deal, just stand by and then read the correct one.

(4) SCREEN FORMAT (H, D, A, N, C)

The **4** key lets you select the format for screen display. After pressing 4, you must press **H, D, A, N** or **C**. Regardless of the format chosen, the status line at the bottom of the screen will apply to all formats. H, D, A, N and C stand for—

H/HEX FORMAT: Displays all 255 bytes as two-digit hex numbers (00-FF without the usual dollar-sign).

D/DECIMAL FORMAT: Displays all 255 bytes as decimal numbers, 000-255. Displaying 255 three-digit numbers on a 960 character screen doesn't leave much spare room, so the numbers are displayed without spaces. An inverse/normal/inverse arrangement separates the numbers visually. The alternative of having the screen scroll, by the way, is not acceptable, so don't mention it.

A/ASCII FORMAT: Displays all 255 bytes as ASCII equivalent characters. Sometimes a byte actually represents a character; sometimes it doesn't. Selecting this option lets you look for words.

N/NO-FLASH FORMAT: Same as A above, but all flashing characters are changed to *inverse* characters, just to make things easier on your eyes. The status display at the bottom will show you the true ASCII character for the byte at the cursor.

C/CATALOG FORMAT: Displays Sector 17's catalog sectors in a mixed hex and ASCII display. Catalog file names are shown as they appear in the catalog. All other bytes are displayed as numbers.

(5) PRINTER DUMP:

Pressing the **5** key will let you dump the current screen information onto your printer. If your printer requires line feeds (if it won't roll the paper up), LET LF$=CHR$(10) at the beginning of the *Byte Zap* listing. If your printer is connected to a slot other than Slot 1, change the variable SS. Be sure and SAVE BYTE ZAP after the changes are made.

(6) CATALOG

The **6** key will catalog your disk so you can compare the catalog to the sectors you are reading. After the catalog, press any key and the sector in memory will be re-displayed.

(7) SLOT/DRIVE CHANGE

The **7** key will let you change the drive that will next be accessed by the program. Hitting RETURN defaults to the slot or drive value printed on the screen. Do not select a drive or slot that is not connected or the program will hang.

(8) DISK MAP

Pressing the **8** key causes the program to read Track 17, Sector 0 (called VTOC or Volume Table of Contents) and display a "Map" showing which sectors on the disk are being used (+) and which are free (.). Compare maps of a newly-initialized disk and an older disk, one with many saved and deleted files. You may dump the disk map to your printer. See Option 5 above if you have problems making your printer work properly.

(9) CHANGE A BYTE

Option **9** lets you change the value for a byte. This change takes place *in memory only* and will not be written to the disk until you write an entire sector to disk. So it's perfectly o.k. to play with this feature. There are three ways to enter a new value for a byte—

HEX: Type a "$" followed by a new value (00-FF) and hit RETURN.

DECIMAL: Type the decimal number (0-255) and hit RETURN.

ASCII: Type an **N** (for Normal) or an **I** (Inverse) or an **F** (Flashing) or an **L** (Lower case) or a **C** (Control) followed by a character. For example, **FG** would insert the value for a Flashing "G" (7) into the byte at the cursor. **I%** would be an inverse "%" (value 37). **NO** is a normal "O" (value 207). **CM** is a control-M (carriage return, value 141). **LA** would be a lower-case "A". A space is entered with the SPACE BAR. To enter a flashing space, for example, enter "F " (F followed by a space, then RETURN). A normal space is "N ".

(See *Free Chart*, page 14, for character values.)

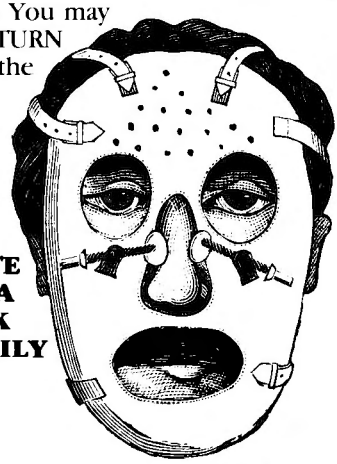
(0) QUIT

Pressing the **zero** key lets you quit *Byte Zap*. To run it again, type "RUN" or "RUN BYTE ZAP".

(—) WRITE SECTOR TO DISK (EQUAL Key on IIe)

Pressing the **hyphen** key will let you write all of the data on the screen onto the disk. Even if you want to change just one byte, you have to write the entire sector. Whatever was on the disk in that sector will be replaced. You may write to the *same* sector being edited by answering **Y** when asked "WRITE TO SAME SECTOR?", or to *another* sector by answering **N**. You may escape this feature by pressing RETURN only. Please heed the warnings on the Key Chart and elsewhere in this documentation regarding writing onto disks.

A WARNING (in Bold Type)
IF YOU ARE GOING TO WRITE DATA ONTO A DISK, MAKE A BACKUP COPY OF THE DISK FIRST. A MISTAKE COULD EASILY BE FATAL TO THE DISK!



Protect yourself.

Some *Byte Zap* Experiments

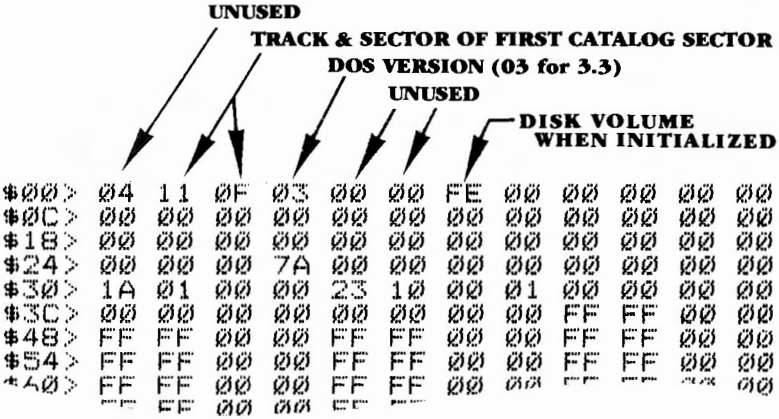
Here are just a few suggestions on how to use *Byte Zap*. As you get into it, I'm sure you'll come up with more. First, a few words about some special areas of Apple disks:

TRACKS 0-2: DOS

Most disks have a copy of Apple's DOS program on Tracks 0-2. When you boot a disk, DOS, actually a machine-language *program*, is transfered from the disk into the memory (RAM) of your Apple. You can look at DOS with *Byte Zap* and even make changes in it, so that when that disk is booted, your DOS changes will be in effect. If you have a blown DOS, you could even transfer a new one, sector by sector, from another disk. Don't waste your time though; use *Master Create* from the *System Master*.

TRACK 17, SECTOR 0: VTOC

Track 17 ($11) Sector 0 ($00) contains a disk's **VTOC** (pronounced *Vee-Talk*) or Volume Table of Contents. Think of the VTOC as a map telling your Apple what is where on the disk. And while you're thinking, picture a disk as a phonograph record. The read/write head corresponds to the phono needle, moving in and out, from track to track as the disk spins at a constant speed. But, unlike a record, programs on a disk are often scattered, starting on this sector and continuing on that one, with the "needle" moving appropriately. The Disk Map feature of *Byte Zap* simply reads the VTOC; Bytes $38-$C3 (56-195). Here is a byte breakdown of the first few bytes of VTOC:



TRACK 17, SECTORS 1-15: THE CATALOG

The rest of Track 17 contains the catalog or *directory* of a disk. Each sector may contain up to 7 file names and pertinent information about each file. There is a 105 file name limit on a 3.3 disk, regardless of the amount of data in each file (7 files × 15 sectors = 105). When you first run *Byte Zap*, you will see Track 17 ($11), Sector 15 ($0F), the first catalog sector (Sector 1, usually unused, is the last).

A TEST

I'm going to initialize (erase & reformat) and create a test disk. If you do the same, you might learn something.

1. If you haven't already done so, boot a *normal* disk, not one that has been "updated" with a program such as *FDOS*. By the way, *PLE* will cause no problem.

2. Type "NEW" (return).

3. Insert a new disk (or a *used* one), and type "INIT HELLO".

4. Load *Byte Zap* and save it on the new disk.

5. Type "LOCK BYTE ZAP".

6. Type "BSAVE BINARY FILE,A$2000,L$1000".

7. Type "SAVE GOOD-BYE".

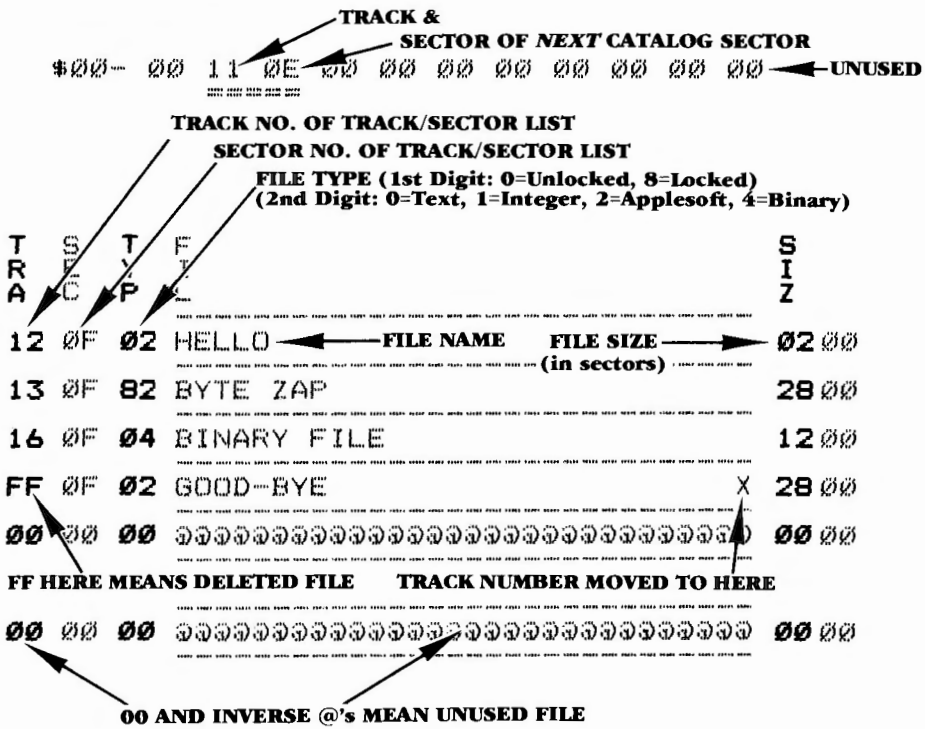
8. Type "DELETE GOOD-BYE".

Your catalog should now look like this:

```
DISK VOLUME 254

 A ØØ2 HELLO
*A Ø4Ø BYTE ZAP
 B Ø18 BINARY FILE
```

Now run *Byte Zap* and look at Track 17, Sector 15. Here is an explanation of what you will see.

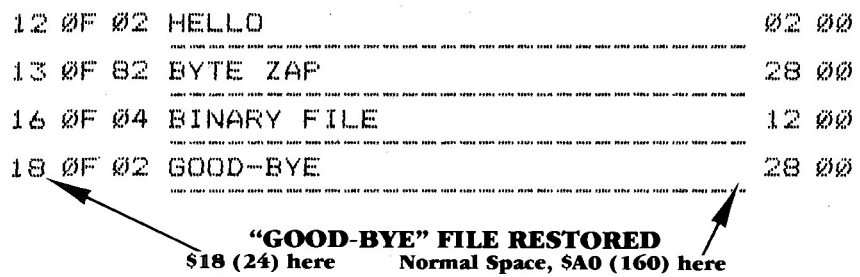


RESTORING A DELETED FILE

Let's undelete our deleted *Good-Bye* file. Notice how the file name doesn't show up when we catalog the disk, but it *does* show up in the sector listing above. The problem is, the *next* file we save will *replace* our *Good-Bye* file name. DOS always looks for the *first available* file name slot. Another problem is that part of our file's data could have been over-written. Well, there's one way to find out—

As you see in the diagram above, our deleted file has had its track number replaced by an FF ($FF or 255). The number that *was* there has been moved to the 30th (last) character in the file name. In this case, it shows as an inverse X. All you have to do to undelete a file is replace the FF with the value of that inverse X (or whatever character you see).

Move the cursor to Byte $94 (148), the 30th character of the 4th file name. Notice in the status display that this inverse X has a value of $18 (24). Now move the cursor to the FF at Byte $74 (116). Change the value by selecting Key Chart Option **9** and entering either **$18** (hex) or **24** (decimal). You could even enter **IX** (for Inverse X). Now move the cursor back to Byte $94, select Option **9**, and change the inverse X to a normal space by entering an "**N** " (N-space). Or you could enter an **$A0** or a **160**. Now the proper changes are in memory. If you catalog the disk by pressing **6**, however, you will find no *Good-Bye* file. That's because the disk doesn't know what we've done. We need to ***write*** our changes onto the disk. Press the "Write Sector to Disk" key and answer **Y** to the question. *Now* catalog, and the file *is* undeleted... probably. TRANSFER THE FILE IMMEDIATELY TO ANOTHER DISK to prevent it from being overwritten.



CUSTOM CATALOG TITLES

Notice the flush left inverse titles in the *Apple Mechanic* catalog. They were made using *Byte Zap* to rename dummy file names. Because of their strange structure, it would be difficult, almost impossible, to *use* these files, but they do make nice labels. Let's rename our *Binary File* in our test catalog. Let's call it *Title* and print the word in inverse.

We'll first need to install 7 *backspaces* to make the name print at the left margin of the screen. Move your cursor to Byte $54 (84). Select Option **9** and enter a **CH** for ctrl-H (backspace). Do this 7 times in Bytes $54-$5A (84-90). Notice how these ctrl-H's show as normal H's in the sector display. The status display in the lower-right of the screen shows their true value by printing a ^H (for ctrl-H). Remember, instead of using CH's, you could have entered the ctrl-H's as **$88**'s or as **136**'s.

```
120F02 HELLO                                          0200
130F82 BYTE ZAP                                       2800
160F04 HHHHHHH TITLE  <--(Inverse "TITLE")            1200
       7 CTRL-H's
       (appear as normal H's)
```

Now enter the inverse file name, *TITLE*, by entering **IT, II, IT, IL** and **IE** in Bytes $5B-$5F (91-95). Write this sector to the disk with the **hyphen** option, and catalog your disk. You've got it! Experiment by injecting such exotic characters as ctrl-M's (carriage returns—$8D/141), ctrl-J's (line feeds—$8A/138), and ctrl-G's (bells—$87/135) into file names.

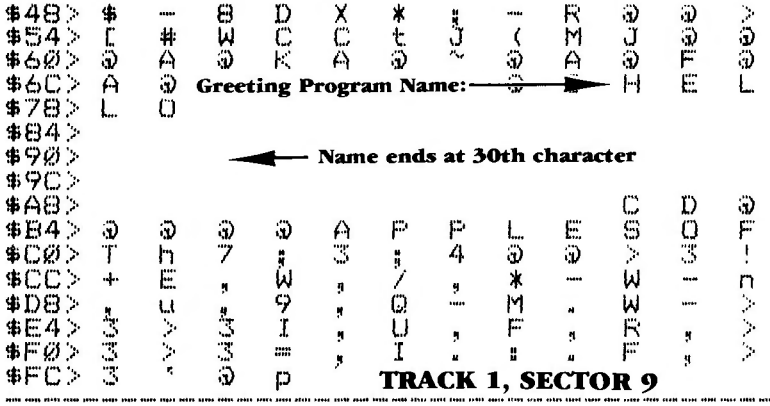
Suppose you wanted a dummy file name to be a string of 30 hyphens, to make a separation in your catalog. Just save a blank file called "A----------". Now use *Byte Zap* to illegally change the A to a hyphen.

Suppose you want a *blank* file name and don't want the sector number and lock code to show in the catalog. Just use *Byte Zap* to rename a file 7 *ctrl-H's* (7 backspaces). Since file names have spaces automatically added to make them 30 characters long, these spaces *erase* the sector number and lock code!

CHANGING A GREETING PROGRAM NAME

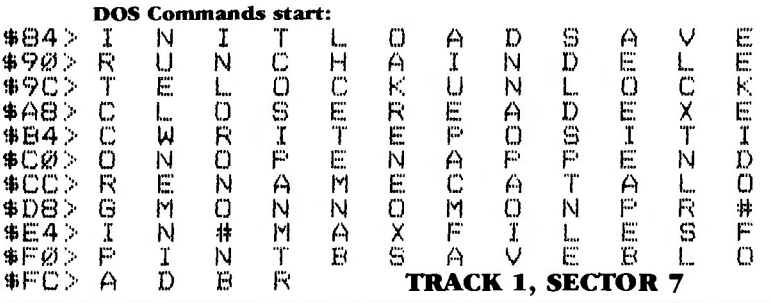
You'll find a disk's greeting program name, the name of the file that is run when the disk is booted, in DOS, on Track 1, Sector 9, Bytes $75-$92 (117-146). Normally, to change this file name, you can simply BRUN *Master Create* from your *System Master* disk, but it won't let you use tricky file names with inverse or other special characters.

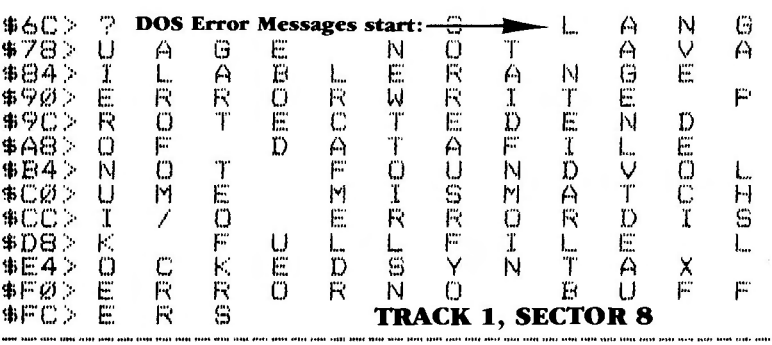
Select Option **2** and enter Track **1**, Sector **9**. If your drive can't read this sector (if it goes *bla-aaa-aa-ttt!*), you probably are using or are trying to read a non-standard DOS. Re-boot if necessary. Once the sector is displayed, you need to change the screen format to ASCII. Select Option **4** and select **A**. Aha!...Words! There's the greeting program name (often *HELLO*) at Byte $75 (117). Change it as described above, and then write this sector to the disk. You also need to create an *identical* file name somewhere in your catalog, Track 17. Now when you boot *this* disk, *that* program will run.



CHANGING COMMANDS & ERROR MESSAGES

DOS's commands and error messages reside on Track 1, Sectors 7-9. Change them as you like. Notice that all of the characters *except* the last character are flashing. This doesn't mean that the characters flash, it means that these are *low-byte* characters. The last character is a *high-byte* character to tell DOS where the command or message ends. Use Option **9** to change these bytes. Just be consistent with the flash and no-flash format that exists. Write the changed sectors back to the disk. Now when you boot *this disk,* your changed commands and error messages will be in effect.





MORE TRICKS

On Track 2, Sector 2, you will find a disk's *Disk Volume* heading (spelled backwards). This is the heading that will appear at the top of a catalog if you boot *this disk*. Change it as you like. And *Barsbait?* Sure, change it too. Those are the file codes that appear in front of your catalog file names. The first B and A and the S are mystery codes, perhaps for future use. R, B, A, I and T stand for Relocatable, Binary, Applesoft, Integer and Text files. Change them to any character; inverse, normal or flashing; that you want.

TRACK 2, SECTOR 2

```
$0C> M  d  S  M  k
$18> P  *  !  +  S
$24> -  @  5  -  T
$30> 5  *  .  @  S
$3C> P  L  -  :  S
$48> +  S  -  @  -
$54> d  S  )  @  -
$60> A  P  "  )  B
$6C> D  P  V  )  E
$78> m  ?  J  )  J
$84> A  @  H  M  E
$90> .  (     L  V
$9C> L  O  @  @  @
$A8> I  A  B  S  R
$B4> O  V     K  S
$C0> @  ^  @  @  @
$CC> @  @  @  @  @
$D8> @  @  @  @  @
$E4> @  @  @  @  @
$F0> P  @  A  @  @
$FC> @  @  @
```

AND MORE

Using your expendible disk (in case you blow it), try messing around with your various catalog numbers.

You can lie about a file's size by changing it to 0 or 255 ($FF); kind of like messing with your car's odometer (only *this* won't wear out your drill motor). Only the first byte (the first pair of hex digits) show up in the catalog (as a 3-digit decimal number). The next two digits (usually zeros) make up the most-significant byte of the size, which doesn't show in the catalog.

Don't change the track and sector numbers to the left of a file name (exception: see *Undeleting Files* above). These tell DOS where the track/sector list is for that file. The track/sector list tells DOS where the program is, sector by sector.

Changing the file type code can be fun. Try changing an 02 (Applesoft) to an 03 (unused), and then run the program. Wait a minute—This is getting dangerous! I'm going to leave the rest of the experimenting up to you.

If you're having fun doing all of this, pick up a copy of *Beneath Apple DOS* at your computer store. Meanwhile, have fun with *Byte Zap!*.

FREE NEWSLETTER

Sign up today; your Beagle Bros Bulletin will be headed for your mailbox tomorrow!

Beagle Bros Bulletin

VOLUME 0, NUMBER 255 "ALL THE NEWS THAT FITS" OCTEMBER 1984

Hello there!

Beagle Bros is out to bring computer reading matter up to the level of the people who use computers. Sign up today for the Beagle Bros Bulletin. It's easy and it's free! Just get your name and address to appear in our mail box. And do it NOW, o.k.? Our trusty mail lady, Flo Chart, will see that you get your copy of the latest Bulletin immediately. Flo says if you've ever received anything from Beagle Bros by mail, you're already on our list and will get your next issue as soon as it's off the press.

We'll be printing a new Bulletin every three months or so. If yours didn't arrive today, stand by; we're working on a new one right now.

Hot TIPS

You'll find tips for beginners, tips for experts, mystery programs, and informative input from our readers. We'll be reporting news about other software on the market, telling you what we like and what we don't like. Keep informed and entertained with the Beagle Bros Bulletin.

There will be a New Peeks & Pokes section in each Bulletin too. Readers have already written us with several additions to our famous chart. We're always out to uncover every Apple secret and get our hands on.

New Products

Keep up-to-date on the latest Beagle Bros disks, tip books and so on. The Bulletin will offer complete descriptions of our products as well as news on product updates and tips on how to make the best use of our programs.

Famous Columnist!

UNCLE LOUIE

We just bought our Uncle Louie a word processor for his Apple, and he's using it too! Some of his tips have been around longer than Louie, but we bet they'll teach you a thing or two. As always, Louie will be sharing his giant collection of "UL Approved" Two-Liners. These contest-winning entries from our readers are truly amazing!

50 Free Charts!

If you belong to an Apple users' group (or have at least 50 neighbors with Apples), we'll treat you to an equal number of Peeks & Pokes charts (and Bulletins) for every name and address we receive. 50 [illegible] offer.

DISCLAIMER: The Beagle Bros Bulletin is nowhere near as big as depicted here. Two men would actually be MUCH bigger than a Beagle Bros Bulletin.

Now Available at your Apple Dealer—

Bert Kersey's
"TYPEFACES for Apple Mechanic"
26 all new shape-fonts for use with
Apple Mechanic's Xtyper and Hi-Writer
programs.

Beagle Bros™
INDOOR SPORTS