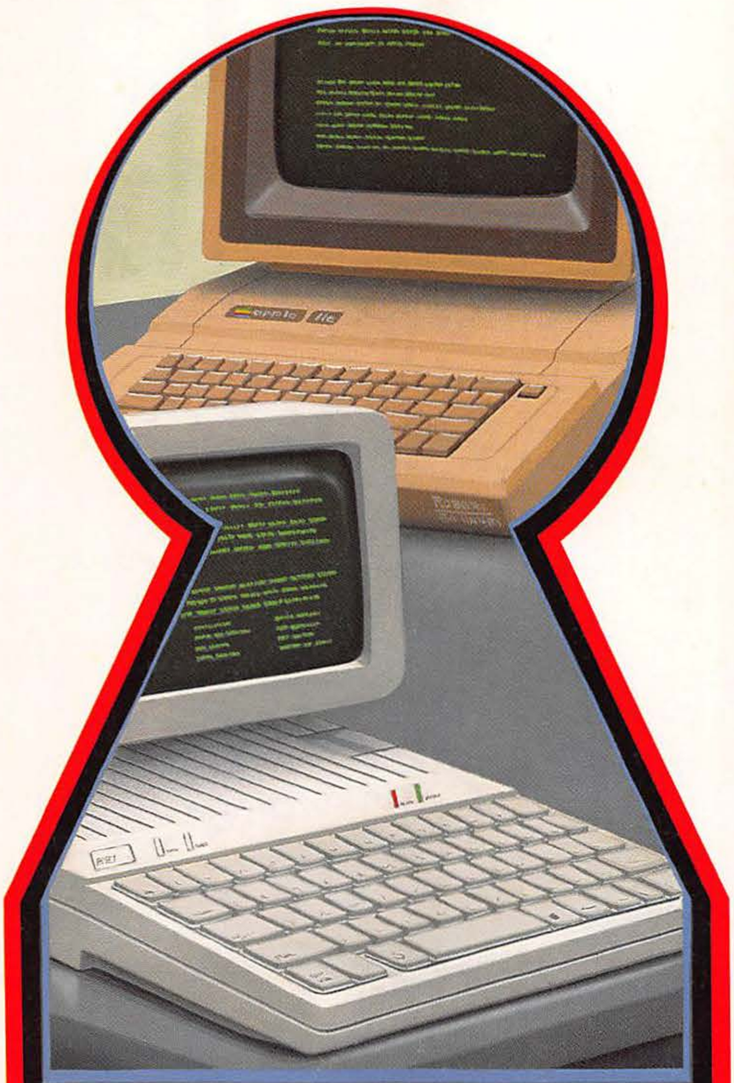
For the Apple II Series

more Apple® Secrets

The Best Tips and Techniques From nibble



The Key to Hidden Apple Treasures

More Apple Secrets

Edited by David Szetela

Nibble Publications
MicroSPARC, Inc.
Concord, MA

Editorial Direction: David Szetela
Technical Direction: David A. Krathwohl
Copy Editors: Mary Locke, Charlotte A. Szetela
Technical Editors: Rich Williams, Owen Linzmayer
Cover: Paul J. Gagnon
Printed and bound by CSA Press, Hudson, MA

ISBN 0-912341-24-8

Library of Congress Number 86-063016

Printed in the United States of America

1 2 3 4 5 6 7 8 9

86 87 88 89 90 91 92

TABLE OF CONTENTS

INTRODUCTION

The Apple II Plus, IIc, IIe and IIGS are full of hidden treasures -- discover them for yourself in this exciting collection of articles and programs.

More Apple Secrets is packed with over 50 of the best Apple Tips and Techniques from Nibble Magazine. Nibble's experts teach you their tricks for creating text windows like those on the Macintosh, automatically converting lower-case letters to upper-case, compressing Hi-Resolution graphics files so that they take up less disk space, and speeding up programs enough to even hear the difference!

Or perhaps you want to know how to add dozens of new colors to your Hi-Resolution palette, or enhance your programs with two-voice music and sound effects. Easy! These programming treasures and many more can be discovered in More Apple Secrets.

If you are new to Apple computing, you'll appreciate the step-by-step instructions for entering and saving programs. It's just like a cookbook for Apple users. (See Appendix A.)

Each Apple Secret is a tested, foolproof method for streamlining your Applesoft and machine language programs, ranging from special programming tips to specific techniques. Whether you're new to computing or a seasoned programmer, this book will provide programs and tips you can use. Most articles include subroutines you can use in your own programs. The authors give detailed, line-by-line explanations of the programs so you can understand the programming logic. And the demonstration programs show you how to use the techniques. You'll find programs that let you:

- Print Hi-Resolution text
- Create text windows like the Apple Macintosh
- Print custom labels
- Explore your Apple's memory
- Customize your favorite DOS command
- Modify Applewriter to work on the IIc
- Learn how the computer sorts information

...and much more.

More Apple Secrets programs are also available on diskette; see the bound-in ordering card for details.

Disk Head Cleaner

Prevent the loss of important programs or data by keeping your 5.25" disk drive heads clean. This DOS 3.3 program makes sure your cleaning kit covers the entire surface.

by Art Mena

To paraphrase a line by Jack Webb in the movie *The D.I.*, "Your disk drive is your friend. If you take care of it, it will take care of you. But if you ever let it down, it will certainly let you down." Your disk drive is about the most essential part of your system. Yet, it might be the cause of many headaches. It has moving parts that wear, and more important, they get dirty. Using a disk drive cleaning kit and the Head Cleaner program, you can keep your drives clean and healthy.

DISK DIRT

If you don't abuse your drive, mechanical wear may never be a problem. However, dirt can cause annoying I/O errors and even destroy the data on your disk. The dirt comes from several sources. One source is dust in the air. However, the disk drive opening is very small and unless your Apple is in a very dirty environment, very little dust enters the drive. The major source of dirt is the disks themselves.

When you insert a disk and close the door to the drive, the disk is pressed against the read/write head by a felt pad. While the disk is rotating, friction between the disk, read/write head and felt pad cause particles of the disk surface to flake off and contaminate the head. Even if you use a high-quality disk, it will still lose its surface over a long period of use. Hard disk drives do not allow the head to touch the disk surface and consequently the surface does not wear. Because all floppy disks will wear, you should always keep backup copies of all of your important programs and data files.

CLEANING KITS AND TIPS

To keep my Apple clean, I always cover it when it's not in use. In addition, I leave the disk drive doors open. If you close the door with no disk in the drive, the felt pad touches the head and may contaminate it. However, in spite of all these precautions, the head will still get dirty.

One of the most important accessories you can have is a disk head cleaning kit. The kit I have contains two cleaning disks and a bottle of cleaning fluid. The cleaning disks are similar to regular disks, except that the magnetic disk has been replaced by a cellulose disk. The fluid is usually isopropyl alcohol, which dissolves the dirt from the head and evaporates entirely, leaving the disk read/write heads clean.

USING A CLEANING KIT

Using a cleaning kit is easy. Squirt some fluid onto a cleaning disk, place the disk in the drive and run the read/write head across the felt pad. I first did this by typing CATALOG. This worked, but only a small portion of the cleaning disk was being used, i.e., the track that DOS thought should contain the directory. Since the cleaning kit is not cheap, I decided to write a program that would run the head between the inner and outer tracks of the cleaning disk in order to use as much of the cleaning surface as possible.

USING THE HEAD CLEANING PROGRAM

The Head Cleaner program (**Listing 1**) starts by asking you which disk drive head you want to clean. You should specify 1 or 2. Pressing 0 will end the program. The Head Cleaner program will run the head back and forth from track $0 to $22 (0-34) four times. This takes about 30 seconds and should be sufficient to clean the head surface.

If you're an average user, you should clean your disk drive's head about once a month to avoid any dirt buildup. This simple act will probably save you much aggravation and may prevent the loss of important programs or data.

ENTERING THE PROGRAM

Type in the Applesoft program shown in **Listing 1**, and save it with the command:

SAVE HEAD.CLEANER

HOW IT WORKS

The program uses the DOS 3.3 RWTS (Read or Write a Track or Sector) routine to move the read/write head across the disk. The RWTS is a set of subroutines that DOS uses to read or write sectors on the disk. However, I do not want to read or write anything, so I use it to find or "seek" the disk tracks.

In order to use the RWTS, I had to set up a table of numbers called the IOCB (Input Output Control Block). This table contains the specific instructions for the routine. The IOCB is contained in the DATA statements in **lines 800-810**. It is stored in memory using the POKE statements immediately following the DATA statements. Since we do not want to read or write anything, we will use the zero (or null) command code to cause the head to go to the desired tracks.

LISTING 1: HEAD.CLEANER

```
10  REM       ***********************
11  REM       *    HEAD.CLEANER     *
12  REM       *     BY ART MENA     *
13  REM       * COPYRIGHT (C) 1983 *
14  REM       * BY MICROSPARC, INC *
15  REM       * CONCORD, MA. 01742 *
16  REM       ***********************
100  REM
110  REM
120  REM  USE THIS PROGRAM WITH
130  REM  A DISK DRIVE HEAD
140  REM  CLEANING KIT TO KEEP
150  REM  YOUR DRIVES CLEAN.
160  REM
170  REM  JUST ENTER "RUN" AND
180  REM  FOLLOW THE DIRECTIONS.
190  REM
200  REM
210 B$ = CHR$ (7) +  CHR$ (7) +  CHR$ (7)
220  GOSUB 820
230  REM
240  REM  GET THE DRIVE NUMBER
250  REM
260  TEXT : HOME : POKE  - 16368,0
270  VTAB 3: HTAB 12: INVERSE : PRINT " HEAD CLEANER ": NORMAL
280  PRINT : PRINT  TAB( 13)"BY ART MENA"
290  PRINT  CHR$ (7)
300  PRINT : PRINT "  ENTER THE DISK DRIVE NUMBER THAT"
310  PRINT "YOU WANT TO CLEAN (1/2,0=END)  ?"
320  PRINT "===> ";: GET DR$
```

```
330 DR =  VAL (DR$)
340  IF DR$ = "0" THEN  TEXT : HOME : END
350  IF DR <  > 1 AND DR <  > 2 THEN  PRINT : PRINT : FLASH :
     PRINT "INCORRECT DISK DRIVE";B$: NORMAL : FOR I = 1 TO
     1000: NEXT I: GOTO 260
360  PRINT DR$: PRINT
370  REM
380  REM  PRINT DIRECTIONS
390  REM
400  PRINT B$: PRINT " PUT SOME CLEANING FLUID ON A 5 1/4"
410  PRINT "INCH CLEANING DISKETTE AND PLACE IT"
420  PRINT "IN DRIVE NUMBER "DR$".  PRESS  ";: INVERSE : PRINT "
     RETURN ": NORMAL
430  PRINT "WHEN YOU ARE FINSIHED."
440  REM
450  POKE  - 16368,0
460  IF  PEEK ( - 16384) <  > 13 AND  PEEK ( - 16384) <  > 141
     THEN  GOTO 460
470  POKE  - 16368,0
480  REM
490  POKE 818,DR
500  POKE 34,10: HOME
510  PRINT : PRINT  CHR$ (7)" RUNNING THE HEAD BACK AND FORTH"
520  PRINT " 4 TIMES.  PRESS ";: INVERSE : PRINT " ESC ";:
     NORMAL : PRINT " TO STOP"
530  PRINT
540  REM
550  REM   USE THE RWTS "SEEK"
560  REM   COMMAND TO RUN THE
570  REM   HEAD ACROSS THE DISK
580  REM
590  FOR CNT = 1 TO 4
600  VTAB 15: PRINT "COUNT ==> ";CNT
610  FOR TRACK = 0 TO 34
620  GOSUB 710
630  NEXT TRACK
640  FOR TRACK = 34 TO 1 STEP  - 1
650  GOSUB 710
660  NEXT TRACK
670  NEXT CNT
680  PRINT : PRINT "ALL DONE !"
690  FOR I = 1 TO 1500: NEXT I: GOTO 260
700  REM
710  REM   HEAD SEEK SUBROUTINE
720  REM
730  IF  PEEK ( - 16384) = 27 OR  PEEK ( - 16384) = 155 THEN
     POP : GOTO 260
740  VTAB 17
750  PRINT "SEEKING TRACK ==> ";TRACK;"   "
760  POKE 820,TRACK
770  CALL 837: REM  CALL RWTS
780  RETURN
790  REM  IOCB FOR RWTS
800  DATA   1,96,1,0,0,0,65,3,0,128,0,0,0,0,0,96
```

```
810  DATA  1,0,0,239,219,160,48,169,3,32,217,3,96
820  RESTORE
830  FOR I = 816 TO 844
840  READ D: POKE I,D
850  NEXT I
860  RETURN
```

DOS 3.3 Fast Load Enhancement

Fast Load Enhancement replaces the standard DOS 3.3 LOADS with a high efficiency Load function that is two to five times the speed of LOAD times.

by Thomas N. Burt

I spend a lot of time doing assembly language and BASIC programming, so I get a little weary waiting for DOS 3.3 to load long programs. Nine months ago, I got a hard disk drive with the intent of "killing the problem with hardware." I was naturally expecting fantastic performance gains. To my dismay, the hard drive was barely twice as fast as the floppy disk drive. Obviously, something was amiss inside of DOS 3.3 itself. Armed with a copy of *Beneath Apple DOS* by Don Worth and Pieter Lechner, the *Apple DOS Reference Manual*, a good disassembler, and specifications for the Apple Disk II drive, I undertook to resolve the problem.

First I'll explain DOS 3.3 I/O performance, why DOS 3.3 is slow and how to speed it up. The program Fast Load Enhancement is a simple modification that you can install as a permanent part of DOS 3.3. The program can speed up the LOAD, BLOAD, RUN and BRUN commands by up to five times over their native DOS 3.3 performance. For example, the time to LOAD a 93-sector BASIC program with standard DOS 3.3 is 23 seconds, versus 5.5 seconds with the Fast Load Enhancement.

DOS 3.3 ORGANIZATION

A Disk II disk has 35 tracks of 16 sectors each. A sector is the smallest unit of information on a disk that can be accessed separately.

Each sector holds 256 bytes of data plus an address header that identifies the track, sector number and volume number. Each track corresponds to a discrete position to which the read/write head of the disk drive can be positioned. As the disk drive turns, each sector on the track rotates under the head and can be read from or written to. Sectors are numbered consecutively around the track from 0-15, in the order of rotation under the read/write heads.

Over this physical sector numbering, DOS 3.3 superimposes a logical numbering as shown in **Figure 1** (values are decimal; hexadecimal equivalents are shown in parentheses).

FIGURE 1: ***DOS 3.3 Logical Sector Numbering***

track logical	track physical	track logical	track physical
15 ($F)	15 ($F)	7 ($7)	1 ($1)
14 ($E)	2 ($2)	6 ($6)	3 ($3)
13 ($D)	4 ($4)	5 ($5)	5 ($5)
12 ($C)	6 ($6)	4 ($4)	7 ($7)
11 ($B)	8 ($8)	3 ($3)	9 ($9)
10 ($A)	10 ($A)	2 ($2)	11 ($B)
9 ($9)	12 ($C)	1 ($1)	13 ($D)
8 ($8)	14 ($ E)	0 ($0)	0 ($0)

Essentially, the logical sector order is descending from 15-0 with the logical sectors on every other physical sector. Sectors 0 and 15 are handled specially so that their physical and logical numbers agree. Two and 1/16 complete turns of the disk are required for all 16 logical sectors to pass under the read/write heads in consecutive order. This descending order logical numbering is important, because that is precisely the order in which DOS reserves the sectors of a track when creating a file.

How Disk Reading Works

The key routine is the DOS Read/Write Track/Sector (RWTS) routine. For any program to read a sector of data, the program makes a machine language subroutine call (JSR) to RWTS, supplying a parameter list containing the desired slot, drive, track, (logical) sector, volume number, and the data's memory address. The DOS 3.3 File Manager is the most common caller of RWTS.

When called, the RWTS first turns the disk drive on, if necessary. Once on, the disk drive keeps turning. Next, the RWTS positions the read/write head to the correct track and then waits for the desired logical sector to rotate under the head. (It looks at the address headers for a match on the corresponding physical sector.)

Then the RWTS transfers 342 bytes of encoded raw data from the disk into a special area of memory. The raw data is decoded into a normal 256-byte sector format and moved to the memory location specified by the caller. The same general sequence occurs on a write, except that the data is moved from the caller's memory location, encoded into raw data, and then transferred to the disk.

As you can imagine, this process takes time — much more time than the brief interval between the end of one physical sector and the beginning of the next. The alternating spacing of logical data sectors on a track allows time for RWTS to decode and move, and for the caller of RWTS to process one sector, and then to resume reading the disk before the start of the next logical sector has rotated past the disk's read/write head. This works great as long as the caller of RWTS doesn't allow too much time to elapse before calling RWTS again.

Disk I/O Timing

A Disk II drive turns at 300 rotations per minute (5 rotations per second). With 16 sectors per track, this rate is equivalent to 80 sectors per second. In reading every other physical sector, the intersector time available for processing is therefore 1/80th of a second, or 12.5 milliseconds. RWTS uses about seven milliseconds of this time itself for raw data decoding and moving. This leaves only about five milliseconds of processing time for the caller of RWTS before the next logical sector of the current rack rotates into position under the read/write head of the disk.

File Space

Files are created one sector at a time. As successive sectors are written, unused disk sectors must be located and reserved before the data is actually transferred. This is necessary to ensure that sectors of other files on the disk are not overwritten by the new file.

As the file is written, the DOS File Manager must remember which sectors were used, and in what order. For each file, the File Manager creates a track/sector list that records the track and logical sector number of each successive sector of the file.

To choose which sector to allocate to a file, the File Manager first finds a track with some unused sectors. Within that track, the File Manager starts looking for free sectors from sector 15 to sector 0. Once a file is started on a given track, all free sectors on that track are allocated to the file until there are none left. Then a new track is selected. Thus, the DOS File Manager is recording successive sectors of a file in exactly the order (15-0) for maximizing the speed of file I/O.

Load Performance

Now let's see how all the above information pertains to the performance of LOADs. The standard DOS 3.3 LOAD function uses the File Manager, in conjunction with RWTS, to copy data between the disk and the Apple's memory. LOAD reads the program file into memory in exactly the same way as a user's program would read a text file. This process involves

OPENing the program file, then calling the File Manager to read it sequentially (as defined by the track/sector list), a sector at a time.

Each sector is read into a DOS file buffer (using the RWTS), and then COPYed a byte at a time from the file buffer to its final destination in memory. As each byte is moved by the File Manager, two separate file position pointers are updated.

Unfortunately, this process of doubly moving the sector a byte at a time, after it has been read in and decoded by RWTS, takes so long (over 25 milliseconds) that the next logical sector on the track has long since rotated past the read/write head and is missed. When RWTS is called by the File Manager to read the next consecutive sector of the program file, it must wait an entire revolution of the disk for that sector to again rotate under the read/write head. Therefore, instead of proceeding at a nominal transfer rate of eight sectors per revolution of the disk, (forty per second) loading proceeds at the pitiful rate of only one sector per revolution (five per second).

A SOLUTION

I reasoned that if the DOS 3.3 load processor took less time to process sectors (so that each successive sector of the program file could be read before it has rotated on by the read/write heads), the effective data transfer rate for LOADs would be eight times faster.

Of course, there is some fixed overhead time in the loading process that is the same regardless of how fast data is read. The file must be OPENed, which involves turning on the disk drive (750 milliseconds) and searching for the file's entry in the disk catalog (minimum 350 milliseconds). When all sectors of a track have been read, the read/write head must be moved to a new track (average 300 milliseconds). Then RWTS must wait an average of half a revolution (100 milliseconds) to locate sector 15 on the new track. With an eight-fold raw data rate increase, the net increase in LOAD times is still a respectable two to five times faster than standard DOS 3.3. Longer files would show the most improvement.

Looking at the entire load process, I questioned the value of first reading the data sector into a separate DOS file buffer and then copying it to some other memory location — all the while maintaining a set of file position pointers that will never be used again. Loading is a very specialized form of file processing, since the entire program file is read or written sequentially into memory as a continuous block.

I needed a way to intercept the File Manager I/O request submitted by the load processor, and then to use a high-efficiency routine to call RWTS and stream the program directly to its final place in memory. Naturally, the program image needed to be preserved correctly under all circumstances and the DOS error handling facilities must still operate properly in the event of an I/O or operator error.

FAST LOAD ENHANCEMENT

The Fast Load Enhancement consists of two parts: a machine language subroutine that installs into DOS 3.3, and a small Applesoft program that creates an EXEC file to install the subroutine and a few other patches. The machine language program is deceptively short and simple. It makes extensive use of existing File Manager internal subroutines to handle the track/sector list and the data transfers.

ENTERING THE PROGRAMS

Please refer to Appendix A for help in entering FLE.OBJ (**Listing 2**). The listing shows ADDRESS:HEX DATA in exactly the form needed for entry via the Monitor's memory entry command. Key the program in starting at address $BEAF and save it with the command:

BSAVE FLE.OBJ,A$BEAF,L$B7

Next, key in the Applesoft program in **Listing 3** and save it with the command:

SAVE INSTALL FILE CREATOR

Then RUN it to create the EXEC file on the same disk as the FLE.OBJ file. Once you have done this, simply boot a copy of DOS 3.3, insert the disk with the various "FLE" files, and EXEC the FLE.EXEC file to install the Fast Load Enhancement.

Make Fast Load Permanent

To capture the modified DOS permanently, take a disk that has already been initialized, but that has no data on it that you need. Type CATALOG to determine the volume number used when the disk was initialized. Either key in or LOAD the desired Hello program. Then enter the command:

```
INIT HELLO,Ss,Dd,Vv
```

where *s* is the slot number, *d* is the drive number and *v* is the volume number of the disk on which the new DOS is to be captured.

The new DOS, a new VTOC and a new catalog will be written to the disk. If you boot from this disk, the new DOS with the Fast Load Enhancement will be used to run the Hello program and to load a RAM card if you have one (or if you have a 64K Apple IIe or Franklin Ace). The new disk will be initialized, but not formatted — hence the need to use a previously formatted disk.

The Fast Load Enhancement subroutine replaces the part of the RWTS used to format blank disks. To protect against errors, the FLE.EXEC installation file disables the RWTS FORMAT (code 4) function by changing the first instruction of the Format routine to a simple CLC (Clear Carry), followed by an RTS (Return from Subroutine) instruction. The INIT function can still be carried out, but it will not format a blank disk. Disks must be preformatted by the INIT command from standard DOS 3.3. These disks can then be reINITed, using the Fast Load Enhancement DOS 3.3. The Volume number on the reINIT should be the same as that used when the disk was preformatted.

HOW IT WORKS

Within the DOS Command Handler, the DOS 3.3. LOAD and BLOAD handlers join at a common processing point called RWR, where the DOS File Manager is called to perform a Read Range function. The FLE.EXEC file replaces this call with a JSR $BEAF, a call to the FLE subroutine.

The program flow is described in the listing comments. Basically, the program first sets up various pointers (FLE-FLE1), and copies the remnant of the first sector from the DOS file buffer where OPEN left it (FLE2-FLE3). It then directly reads whole sectors until there are less than 256 bytes left to read (FLE3-FLE8). The last sector, if any, is read into the file buffer and its active part is then moved a byte at a time (FLE8-FLE11).

Finally, there are some special cases to be handled: program file length that is less than 256 bytes, and program length that is an exact multiple of 256 bytes. File Manager subroutines are used to call RWTS, to locate the track/sector list buffer and data buffer, and to handle I/O errors.

COMPATIBILITY

The Fast Load Enhancement should install into any reasonably standard version of DOS 3.3. It has been extensively tested in a BASIC environment. A large family of other machine language processors and utility programs have also been run successfully. Programs that follow standard Apple DOS interface specifications should not have problems.

Most problems with compatibility arise with utilities and special processors written in machine language which may themselves modify DOS 3.3. Programs on copy-protected disks probably will not work since they normally require their own nonstandard versions of DOS. Also, users who have various nonstandard drives, such as eight-inch floppies or hard drives should be careful, since these devices usually have different associated DOS 3.3 support modifications.

Before doing anything irrevocable, like running your production payroll programs with the Fast Load Enhancement, make a backup, and do a test first.

LISTING 2: FLE.OBJ

```
;
;                FLE.OBJ
;   DOS 3.3 LOAD/BLOAD  EXTENSIONS
;      BY TOM BURT, IRVINE CA
;
;        COPYRIGHT (C) 1982
;        BY MICROSPARC INC.
;        CONCORD, MA  01742
;        ALL RIGHTS RESERVED
;
; FLE IS INVOKED WHEN A LOAD COMMAND
; IS PERFORMED.  THE CODE DOES A HIGH
; SPEED LOAD BY GOING DIRECTLY TO THE
; RWTS ROUTINE FOR WHOLE SECTORS.
;
; THIS VERSION INTERFACES TO THE
; STANDARD 48K DOS 3.3., OVERLAYING
; THE FORMAT FUNCTION OF RWTS.
;
;    DOS EQUATES NEEDED FOR ACCESS
;
FMPL      EQU $B5BB          ;FILE MANAGER P-LIST
FMWA      EQU $B5D1          ;FILE MANAGER WORK AREA
STKSAVE   EQU $B39B
FILPOSN   EQU $B5E4          ;FILE POSITION
IOBORG    EQU $B7E8          ;DOS IOB ADDRESS
IOBBUFA   EQU IOBORG+8       ;BUFFER ADDRESS
BUFADR    EQU $42            ; - $43
STBUF     EQU $48            ;-$49  BORROWED FROM RWTS
;
;  FILE MANAGER INTERNAL SUBROUTINES
;
RWTSDRVR EQU $B052           ;RWTS DRIVER
SELDABF   EQU $AF10          ;GET DATA BUFFER ADDR
SELTSBF   EQU $AF0C          ;GET T/S LIST ADDR
RDTSLIST EQU $AF5E           ;READ T/S LIST
SETERROR EQU $B385           ;FILE MGR ERROR EXIT
;
FORMDSK   EQU $BE0D          ;RWTS FORMAT RTN
RWR2      EQU $A40A          ;RWR2 JUMP IN DOS CDI
;
          ORG $BEAF          ;ORIGIN OF RWTS FORMAT
;
;   WORKING STORAGE FOR FAST LOADER
;
```

```
                    FL.OFFST EQU FILPOSN+2   ;OFFSET (2,4)
                    FL.FWA   EQU FMPL+8      ;FWA OF LOAD
                    FL.LDLN  EQU FMPL+6      ;LENGTH OF LOAD
                    ;
                    ; LOAD/BLOAD ENTRY - CALLED AFTER THE
                    ; FIRST 2 OR 4 BYTES HAVE BEEN READ
                    ;
BEAF  D8            FLE      CLD             ;ENSURE RIGHT ARITH MODE
BEB0  BA                     TSX            ;SAVE STACK PTR FOR DOS EXIT
BEB1  8E 9B B3               STX STKSAVE
                    ;
BEB4  AD C3 B5      FLE1     LDA FL.FWA      ;MOVE START ADDRESS
BEB7  38                     SEC
BEB8  ED E6 B5               SBC FL.OFFST    ;BACK UP BY OFFSET
BEBB  8D C3 B5               STA FL.FWA
BEBE  AD C4 B5               LDA FL.FWA+1
BEC1  E9 00                  SBC #0
BEC3  8D C4 B5               STA FL.FWA+1    ;FWA/FWA+1 = FWA-OFFST
                    ;
BEC6  A9 0E                  LDA #$E        ;OFFSET TO 2ND T/S LIST PAIR
BEC8  8D 65 BF               STA TSPTR
BECB  20 10 AF               JSR SELDABF     ;POINT TO DATA BUFFER
                    ;
BECE  AC E6 B5               LDY FL.OFFST    ;GET OFFSET (2 OR 4)
BED1  AD C1 B5               LDA FL.LDLN     ;GET LENGTH (LSB)
BED4  18                     CLC
BED5  6D E6 B5               ADC FL.OFFST
BED8  8D C1 B5               STA FL.LDLN     ;ADD OFFSET
BEDB  A9 00                  LDA #0
BEDD  6D C2 B5               ADC FL.LDLN+1   ;(MSB)
BEE0  8D C2 B5               STA FL.LDLN+1
                    ;
BEE3  F0 5F                  BEQ FLE9        ;SKIP < $100 BYTES LEFT
                    ;
                    ; COPY THE REST OF THE FIRST SECTOR
                    ; FROM THE I/O BUFFER TO MEMORY
                    ;
BEE5  20 5A BF               JSR SETSTBF     ;SET UP STORE BUFFER
BEE8  B1 42         FLE2     LDA (BUFADR),Y  ;FETCH BYTE
BEEA  91 48                  STA (STBUF),Y
BEEC  C8                     INY
BEED  D0 F9                  BNE FLE2        ;KEEP GOING
                    ;
                    ; WE HAVE MOVED THE REST OF THE FIRST
                    ; SECTOR TO THE PROPER PLACEIN MEMORY
                    ; NOW WE READ WHOLE SECTORS UNTIL LESS
                    ; THAN A SECTOR OF DATA IS LEFT.
                    ;
BEEF  EE C4 B5      FLE3     INC FL.FWA+1    ;UP FWA BY $100
BEF2  20 0C AF               JSR SELTSBF     ;POINT TO T/S LIST
BEF5  AC 65 BF               LDY TSPTR
BEF8  D0 08                  BNE FLE6        ;SKIP IF NOT AT END
BEFA  38                     SEC             ;FLAG NEXT T/S LIST
BEFB  20 5E AF               JSR RDTSLIST
BEFE  B0 55                  BCS FLEERR      ;SKIP IF ERROR
BF00  A0 0C                  LDY #$C         ;OFFSET TO FIRST T/S PAIR
                    ;
BF02  B1 42         FLE6     LDA (BUFADR),Y
```

```
BF04  AA                 TAX              ;TRACK
BF05  C8                 INY
BF06  B1 42              LDA (BUFADR),Y
BF08  C8                 INY
BF09  8C 65 BF           STY TSPTR
BF0C  A8                 TAY
BF0D  CE C2 B5           DEC FL.LDLN+1    ;DECR LENGTH BY $100
BF10  F0 15              BEQ FLE8         ;SKIP < $100 BYTES LEFT
                  ;
BF12  AD C3 B5           LDA FL.FWA       ;SET ADDRESS FOR RWTS
BF15  8D F0 B7           STA IOBBUFA
BF18  AD C4 B5           LDA FL.FWA+1
BF1B  8D F1 B7           STA IOBBUFA+1
BF1E  A9 01              LDA #1           ;READ CODE
BF20  20 52 B0           JSR RWTSDRVR     ;READ THE SECTOR
BF23  90 CA              BCC FLE3         ;NO ERROR
BF25  B0 2E              BCS FLEERR       ;QUIT IF ERROR
                  ;
                  ; WE ARE HERE TO READ THE LAST SECTOR
                  ; INTO THE DOS I/O BUFFER AND THEN
                  ; COPY THE REMNANT TO THE END OF THE
                  ; THE CODE IMAGE IN MEMORY.
                  ;
BF27  AD C1 B5    FLE8   LDA FL.LDLN      ;CHECK LENGTH LSB
BF2A  F0 27              BEQ FLE11        ;QUIT IF NO DATA LEFT
BF2C  8A                 TXA              ;SAVE X
BF2D  48                 PHA
BF2E  20 10 AF           JSR SELDABF      ;GET DATA BUFFER ADDRESS
BF31  A5 42              LDA BUFADR       ;SET UP THE IOB
BF33  8D F0 B7           STA IOBBUFA
BF36  A5 43              LDA BUFADR+1
BF38  8D F1 B7           STA IOBBUFA+1
                  ;
BF3B  68                 PLA              ;RECOVER X-REG
BF3C  AA                 TAX
BF3D  A9 01              LDA #1           ;READ CODE
BF3F  20 52 B0           JSR RWTSDRVR
BF42  A0 00              LDY #0           ;SET UP OFFSET INTO SECTOR
                  ;
                  ; WE ARE HERE WITH THE LAST PART OF
                  ; THE DATA IN THE DOS I/O BUFFER
                  ;
BF44  20 5A BF    FLE9   JSR SETSTBF      ;SET UP STORE BUFFER
BF47  CC C1 B5    FLE10  CPY FL.LDLN      ;ALL DATA MOVED?
BF4A  B0 07              BCS FLE11        ;IF YES, QUIT
BF4C  B1 42              LDA (BUFADR),Y   ;FETCH BYTE
BF4E  91 48              STA (STBUF),Y    ;STORE BYTE
BF50  C8                 INY              ;ADVANCE POINTER
BF51  D0 F4              BNE FLE10
                  ;
                  ; ALL DONE WITH THE LOAD
                  ;
BF53  18          FLE11  CLC              ;EXIT CARRY CLEAR
BF54  60                 RTS
                  ;
BF55  A9 08       FLEERR LDA #8           ;I/O ERROR
BF57  4C 85 B3           JMP SETERROR     ;EXIT TO FMNGR
                  ;
```

```
BF5A  AD C3 B5  SETSTBF  LDA FL.FWA
BF5D  85 48              STA STBUF
BF5F  AD C4 B5           LDA FL.FWA+1
BF62  85 49              STA STBUF+1
BF64  60                 RTS
                ;
BF65  00        TSPTR    DFC 0            ;T/S LIST POINTER
                LENFLE   EQU .-FLE

000  ERRORS

BEAF  HEX START OF OBJECT
BF65  HEX END OF OBJECT
00B7  HEX LENGTH OF OBJECT
94F1  HEX END OF SYMBOLS
```

LISTING 3: INSTALL FILE CREATOR

```
100  REM ********************************
101  REM *    FAST LOAD ENHANCEMENT     *
102  REM *     INSTALL FILE CREATOR     *
103  REM *         BY TOM BURT          *
104  REM *    COPYRIGHT (C) 1983 BY     *
105  REM *       MICROSPARC INC.        *
106  REM *      CONCORD,MA 01742        *
107  REM *     ALL RIGHTS RESERVED      *
108  REM ********************************
110 D$ =  CHR$ (4)
120  REM  OPEN THE TEXT FILE
130  PRINT D$;"OPEN FLE.EXEC"
140  PRINT D$;"WRITE FLE.EXEC"
150  REM  LOAD THE FLE SUBROUTINE
160  PRINT "BLOAD FLE.OBJ,A$BEAF"
170  REM  CHANGE LOAD/BLOAD TO CALL FLE
180  PRINT "CALL -151"
190  PRINT "A40B:AF BE"
200  REM  DISABLE RWTS FORMAT (CLC,RTS)
210  PRINT "BE0D:18 60"
220  REM  RETURN TO BASIC
230  PRINT "3D0G"
240  PRINT D$;"CLOSE FLE.EXEC"
250  END
```

RAM-PAD

If you are tired of searching through your manuals and program listings for often-used addresses and routines, RAM-PAD is the utility for you. With it, data may be saved and reviewed at the touch of a key.

by Sandy Mossberg

USING RAM-PAD

RAM-PAD works under DOS 3.3 on an Apple II Plus with a RAM card in slot 0, or on an Apple IIe, IIc, IIGS or Franklin Ace. Typing BRUN RAM-PAD installs the program in the extra 16K of firmware or hardware. I'll use the term "RAM card" or "card" to refer to this bank-switched memory, regardless of whether it resides on a card in slot 0 or is built in.

Four screens (pages A-D) of text may be saved on the RAM card or restored to the display. To save the current screen contents (text page 1), press <CTRL>S followed by the control character representing the desired page (<CTRL>A, <CTRL>B, <CTRL>C or <CTRL>D).

To review the contents of a page, press <CTRL>R followed by the control character representing the desired page. After examining the restored page, pressing <CTRL>X makes the restoration permanent, and places the cursor on the bottom row of the screen. Pressing any other key returns the original screen to view. The flashing cursor disappears after using one of the two main commands. Requesting an invalid page letter evokes a beep, and the cursor reappears. The <RESET> key functions normally.

Since the pages are saved to RAM, turning off the computer destroys the saved pages. PAGE-A for RAM-PAD (Listing 2) may be used to load the pages of RAM-PAD. This Applesoft program provides you with one page of meaningful data that is transferred automatically to page A of the installed RAM-PAD.

ENTERING THE PROGRAMS

Please refer to Appendix A for help in entering **Listing 4**. If you key it in from the Monitor, save it to disk with the command:

BSAVE RAM.PAD,A$8000,L$126

To enter PAGE.A for RAM-PAD, key in **Listing 5** and save it with the command:

SAVE PAGE.A

HOW THE PROGRAMS WORK

RAM-PAD

If the RAM card program does not extend above $F800, it is simpler to place the F8 Monitor ($F800-$FFFF) onto the card so that Monitor subroutines can be called without the need for vectors. I usually opt for a stand-alone machine language program — one that requires no Applesoft or EXEC file interface. To this end, the program must first be loaded into low RAM (below $C000), and then moved onto the card (high RAM). As with all co-resident programs that control the I/O hooks, a special routine should restore these hooks when the <RESET> key disconnects them. These functions are performed in **lines 40-107** and **lines 234-238** of **Listing 4**.

The routines that transfer control from the card (**lines 197-204**) ensure that the immediate mode at entry (Applesoft or Monitor) is preserved.

Lines 250 and 251 and **lines 255 and 256** take advantage of the ability of the Merlin Assembler (published by Roger Wagner Publishing) to assemble a program with more than one origin address. These four equates simply identify the starting and ending low RAM addresses of the vector table and the RAM card program.

Bank Select Switches

Locations $C080-$C08F (bank select switches) are used to manipulate the read/write status of card RAM and motherboard ROM.

Since we will use Bank 2 of the $D000-$DFFF space, only locations $C084-$C087 are pertinent. Since $C080-$C083 is identical to $C084-$C087, only the former will be considered.

For the purposes of the current program, one reference to $C081 turns ROM on and the card off, and two or more successive references to $C081 turn ROM on and write-enable the card. One reference to $C083 read-enables the card and disables ROM, and two or more successive references to $C083 read- and write-enable the card and turn ROM off.

The RAM card program must be accessed by a routine in low RAM that employs one of the switches noted above. This vector (**lines 108-110**) thus becomes the true input handler, and its address may be found in $AA55-$AA56 of the DOS Main Routines Table. When exiting the RAM card, a low RAM vector must also be employed (**lines 114-119**). I have tucked these important vectors into a 33-byte free space within DOS ($BCDF-$BCFF).

The actual RAM card program is reasonably simple. Prior to saving the entry row (**lines 135-136**), keyboard input is filtered. If a valid command does not occur, control returns to the caller (**lines 132 and 134**). If an appropriate command is encountered but does not occur immediately to the right of the Applesoft prompt, RAM-PAD is bypassed (**lines 140 and 144**). If the command is sustained, $94 is subtracted from the ASCII value of <CTRL>R ($92) or <CTRL>S ($93) and stored in the Y-Register (**lines 137 and 145-147**). When (Y) is later incremented (**line 162**), it contains a positive value if the SAVE command were given, and a negative value if RESTORE was invoked (**line 163**).

The Display

The page letter to be saved or restored next must be obtained. On the Apple II Plus, KEYIN places no cursor on the screen, however on the Apple IIe and IIc, a flashing checkerboard is displayed.

Since I prefer no cursor and want the effect to be the same regardless of which machine is used, a mini-KEYIN subroutine is employed (**lines 226-230** to fetch a letter without a cursor prompt. Any character other than <CTRL>A, <CTRL>B, <CTRL>C or <CTRL>D is rejected (**lines 152-156**). The starting location of a valid page is found (**lines 157-158**) by indexing the table in **line 242**.

Memory Move

The subroutine that moves memory (**lines 208-222**) picks up the high-order bytes of the origin (Y-Register) and destination (A-Register) and affects the movement of exactly $400 bytes (indexed by the X-Register).

PMOVE is called by SAVE (**lines 167-169**) to transfer the screen contents to the RAM card, and by RESTORE (**lines 177-193**) to store the current screen, display a page on the card, and restore the original page if any character other than <CTRL>X is typed.

PAGE-A for RAM-PAD

Lines 120-200 and **510-520** format the screen. With ROM read and RAM card write in effect (**line 220**), **line 230** equates A$ to the Monitor command which moves the System Monitor onto the card. The S.H. Lam subroutine (**line 410**) does the following:

1. Adds "PLA PLARTS" (the code at $D9C6) to the Monitor command.
2. POKES the entire command into the input buffer.
3. Clears the Status register.
4. Calls the Monitor Command Processor to execute the command.

The three-byte code added in number 1 allows return to a running BASIC program. **Line 240** disables the card, and **line 250** ends the program. You may eventually want to expand this program to fill all four pages of the RAM-PAD.

Using Monitor commands in an Applesoft program is a valuable asset, and the Lam code is the most efficient method I have found for doing this.

MODIFICATIONS

To provide four more storage pages, add E8ECF0F4 to **line 244** of **Listing 4**, and change **line 157** to CMP #8.

It would be convenient to have RAM-PAD write pages of data into a binary file that could be loaded at any time. The File Manager can be employed for this task; additional commands are necessary. Since the RAM-PAD filters all input, you can easily add handy commands that provide catalog control, entry to the System Monitor, cursor manipulation and other features.

LISTING 4: RAM.PAD

```
1     ***************************
2     *                         *
3     *         RAM.PAD         *
4     *                         *
5     *   by Sandy Mossberg     *  Merlin Assembler
6     * (C) 1983 MICROSPARC INC.*
7     ***************************
8
9     *---------
10    * EQUATES:
11    *---------
12    CH        =    $24       ;Cursor column
13    CV        =    $25       ;Cursor row
14    BASL      =    $28       ;Left margin of current line
15    PROMPT    =    $33       ;Prompt character
16    CVSAV     =    $34       ;Save CV
17    KSWL      =    $38       ;Input hook
18    KSWH      =    $39
19    A1L       =    $3C       ;Start of segment moved
20    A1H       =    $3D
21    A2L       =    $3E       ;End of segment moved
22    A2H       =    $3F
23    A4L       =    $42       ;Destination of move
24    A4H       =    $43
25    DOSWRM    =    $3D0      ;Warmstart DOS
26    SOFTEV    =    $3F2      ;RESET vector
27    KEY       =    $C000     ;Keyboard input
28    STROBE    =    $C010     ;Keyboard strobe
29    TABV      =    $FB5B     ;Set row in (A)
30    SETPWRC   =    $FB6F     ;Set power-up byte
31    KEYIN     =    $FD1B     ;Get keypress
32    MOVE      =    $FE2C     ;Move memory (Y=0)
33    BELL      =    $FF3A     ;Beep!
34    MONZ      =    $FF69     ;Enter system monitor
35    *-------
36    * SETUP:
37    *-------
38              ORG  $8000
```

```
                    * Move F8 monitor ROM to card:
8000: A0 F8                 LDY  #$F8
8002: 84 3D                 STY  A1H
8004: 84 43                 STY  A4H
8006: A0 FF                 LDY  #$FF
8008: 84 3E                 STY  A2L
800A: 84 3F                 STY  A2H
800C: C8                    INY
800D: 84 3C                 STY  A1L
800F: 84 42                 STY  A4L
8011: AD 81 C0              LDA  $C081        ;ROM read, card write
8014: AD 81 C0              LDA  $C081
8017: 20 2C FE              JSR  MOVE
                    * Move program to card:
801A: A0 80                 LDY  #>PGMSTART
801C: 84 3D                 STY  A1H
801E: A0 7D                 LDY  #PGMSTART
8020: 84 3C                 STY  A1L
8022: A0 81                 LDY  #>PGMEND
8024: 84 3F                 STY  A2H
8026: A0 25                 LDY  #PGMEND
8028: 84 3E                 STY  A2L
802A: A0 D0                 LDY  #>RCSTART
802C: 84 43                 STY  A4H
802E: A0 00                 LDY  #RCSTART
8030: 84 42                 STY  A4L
8032: 20 2C FE              JSR  MOVE
                    * Move vectors into DOS:
8035: A0 80                 LDY  #>VECSTART
8037: 84 3D                 STY  A1H
8039: A0 5F                 LDY  #VECSTART
803B: 84 3C                 STY  A1L
803D: A0 80                 LDY  #>VECEND
803F: 84 3F                 STY  A2H
8041: A0 7C                 LDY  #VECEND
8043: 84 3E                 STY  A2L
8045: A0 BC                 LDY  #>VRESET
8047: 84 43                 STY  A4H
8049: A0 DF                 LDY  #VRESET
804B: 84 42                 STY  A4L
804D: A0 00                 LDY  #0
804F: 20 2C FE              JSR  MOVE
                    * Set page 3 reset vector:
8052: A0 DF                 LDY  #VRESET
8054: 8C F2 03              STY  SOFTEV
8057: A0 BC                 LDY  #>VRESET
8059: 8C F3 03              STY  SOFTEV+1
805C: 20 6F FB              JSR  SETPWRC
                    * Set input hook (fall into VRESET):
```

```
                LORAM1   =    *
                *--------------
                * VECTOR TABLE:
                *--------------
                         ORG  $BCDF

                * To RAMCARD:

BCDF: 8D 83 C0  VRESET   STA  $C083        ;Card read
BCE2: 4C 99 D0           JMP  RESET
BCE5: 8D 83 C0  VINPUT   STA  $C083        ;Card read/write
BCE8: 8D 83 C0           STA  $C083
BCEB: 4C 00 D0           JMP  RCSTART

                * From RAMCARD:

BCEE: 8D 81 C0  VEND     STA  $C081        ;Disable card
BCF1: F0 03              BEQ  VMON
BCF3: 4C D0 03           JMP  DOSWRM
BCF6: 4C 69 FF  VMON     JMP  MONZ
BCF9: 8D 81 C0  VRTS     STA  $C081        ;Disable card
BCFC: 60                 RTS

                LORAM2   =    *
                *-----------------
                * RAMCARD PROGRAM:
                *-----------------
                         ORG  $D000


                * Get SAVE/RESTORE command and test validity:

D000: 20 1B FD  RCSTART  JSR  KEYIN
D003: C9 94              CMP  #"S"+1-$40 ;CTL-S (save)
D005: B0 34              BCS  RTS1
D007: C9 92              CMP  #"R"-$40   ;CTL-R (restore)
D009: 90 30              BCC  RTS1
D00B: A4 25              LDY  CV
D00D: 84 34              STY  CVSAV      ;Save entry row
D00F: AA                 TAX
D010: A4 24              LDY  CH
D012: C0 01              CPY  #1         ;Command in column 1
D014: D0 25              BNE  RTS1
D016: 88                 DEY
D017: B1 28              LDA  (BASL),Y
D019: C5 33              CMP  PROMPT     ;Prompt in column 0
D01B: D0 1E              BNE  RTS1
D01D: 8A                 TXA
D01E: E9 94              SBC  #$94       ;R=$FE, S=$FF
D020: A8                 TAY

                * Get page letter command and test validity:

D021: 20 8D D0           JSR  PKEYIN
D024: 38                 SEC
D025: E9 81              SBC  #"A"-$40   ;CTL-A,B,C,D=0,1,2,3
D027: 30 38              BMI  ERROR
```

```
D029: C9 04              CMP  #4
D02B: B0 34              BCS  ERROR
D02D: AA                 TAX             ;Index to page table
D02E: BD A4 D0           LDA  PAGETBL,X  ;Starting address of page

               * Direct flow to SAVE or RESTORE:

D031: C8                 INY             ;R=$FF, S=0
D032: 30 0A              BMI  RESTORE

               * SAVE current text page:

D034: A0 04              LDY  #4
D036: 20 72 D0           JSR  PMOVE
D039: F0 29              BEQ  ENDOLD     ;Always

               * Vector RTS to low RAM:

D03B: 4C F9 BC  RTS1     JMP  VRTS

               * RESTORE page for viewing:

D03E: 48        RESTORE  PHA
D03F: A9 D4              LDA  #$D4
D041: A0 04              LDY  #4
D043: 20 72 D0           JSR  PMOVE      ;Save current text page
D046: 68                 PLA
D047: A8                 TAY
D048: A9 04              LDA  #4
D04A: 20 72 D0           JSR  PMOVE      ;Display designated page
D04D: 20 8D D0           JSR  PKEYIN     ;Pause for keypress
D050: C9 98              CMP  #"X"-$40   ;CTL-X
D052: F0 09              BEQ  R1
D054: A0 D4              LDY  #$D4       ;Recover original page
D056: A9 04              LDA  #4
D058: 20 72 D0           JSR  PMOVE
D05B: F0 07              BEQ  ENDOLD     ;Always
D05D: A9 16     R1       LDA  #$16       ;Restored page remains
D05F: D0 07              BNE  ENDNEW     ;Always

               * Ending routines:

D061: 20 3A FF  ERROR    JSR  BELL       ;Erroroneous page letter
D064: A4 34     ENDOLD   LDY  CVSAV      ;Recover original page
D066: 88                 DEY
D067: 98                 TYA
D068: 20 5B FB  ENDNEW   JSR  TABV       ;Restored page remains
D06B: A5 33              LDA  PROMPT
D06D: C9 AA              CMP  #"*"
D06F: 4C EE BC           JMP  VEND       ;Vector end to low RAM

               * Move up or down:

D072: 85 43     PMOVE    STA  A4H        ;Destination hi
D074: 84 3D              STY  A1H        ;Origin hi
D076: A2 04              LDX  #4         ;$400 bytes to be moved
D078: A0 00              LDY  #0
D07A: 84 42              STY  A4L        ;Destination lo
```

```
D07C: 84 3C        STY  A1L            ;Origin lo
D07E: B1 3C   PM1     LDA  (A1L),Y        ;Get from here
D080: 91 42        STA  (A4L),Y        ;Put here
D082: C8           INY
D083: D0 F9        BNE  PM1
D085: E6 3D        INC  A1H            ;Bump pointers hi
D087: E6 43        INC  A4H
D089: CA           DEX
D08A: D0 F2        BNE  PM1            ;More bytes to move
D08C: 60           RTS                 ;Exactly $400 bytes moved
              * Get keypress (no prompt):
D08D: 2C 00 C0 PKEYIN  BIT  KEY
D090: 10 FB        BPL  PKEYIN
D092: AD 00 C0     LDA  KEY
D095: 2C 10 C0     BIT  STROBE
D098: 60           RTS
              * RESET handler:
D099: A9 E5   RESET   LDA  #VINPUT        ;Restore input hook
D09B: 85 38        STA  KSWL
D09D: A9 BC        LDA  #>VINPUT
D09F: 85 39        STA  KSWH
D0A1: 4C EE BC     JMP  VEND           :Vector end to low RAM
              * RAMCARD page table:
D0A4: D8 DC E0 PAGETBL HEX  D8DCE0E4
D0A7: E4
D0A8: 00      RCEND   HEX  00
              *-----------------------------------
              * Equates in BLOADed (LORAM) program:
              *-----------------------------------
              * Vector table:
              VECSTART =     LORAM1
              VECEND   =     LORAM1+LORAM2-VRESET-1
              * RAMCARD program:
              PGMSTART =     VECEND+1
              PGMEND   =     PGMSTART+RCEND-RCSTART

--End assembly, 294 bytes, Errors: 0
```

LISTING 5: PAGE.A

```
100  REM  *** PAGE.A FOR RAM.PAD ***
110  REM  *** PRINT INFORMATION
120  HOME : FOR I = 1 TO 13: IF I = 8 THEN  PRINT : PRINT
```

```
130  READ A$,B$: PRINT A$;: HTAB (11 -  LEN (B$)): PRINT B$:
     NEXT I: PRINT
140  FOR I = 1 TO 15: READ A$,B$: VTAB I: HTAB 19: PRINT A$;:
     HTAB (33 -  LEN (B$)): PRINT B$: NEXT : PRINT
150  PRINT "BANK2  ROM  CARD  BANK1"
160  PRINT "-----  ---  ----  -----"
170  PRINT "49280   -   R,WP  49288"
180  PRINT "49281   R   W2    49289"
190  PRINT "49282   R   WP    49290"
200  PRINT "49283   -   R,W2  49291"
210  REM           MOVE INFO TO RAM-PAD
220  POKE 49281,0: POKE 49281,0: REM ROM READ, CARD WRITE
230 A$ = "D800<400.7FFM": GOSUB 400: REM EXECUTE MONITOR COMMAND
240  POKE 49281,0: REM DISABLE CARD
250  END
400  REM           S H LAM SUBROUTINE
410 A$ = A$ + " N D9C6G": FOR I = 1 TO  LEN (A$): POKE 511 + I,
     ASC ( MID$ (A$,I,1)) + 128: NEXT : POKE 72,0: CALL  - 144:
     RETURN
500  REM           DATA
510  DATA LEFT,32,WIDTH,33,TOP,34,BOTTOM,35,CH,36,CV,37,PC,58,
     START,103,LOMEM,105,ARRAY,107,FREE-
     1,109,STRING,111,HIMEM,115
520  DATA DOSWARM,976,DOSCOLD,979,DOSHOOK,1002,RESET,1010,AMPER,
     1013,CTLY,1016,KEY,49152,STROBE,49168,SPEAKER,49200,BS,-
     1008,UP,-998,CLREOP,-958,KEYIN,-741,SETKBD,-375,SETVID,-
     365,BELL,-198,MON,-155,MONZ,-151
```

TAB XY

TAB XY is an ampersand utility that provides a shorthand method for positioning the cursor on the screen.

by S. Scott Zimmerman

TAB XY lets you avoid having to constantly type statements such as HTAB 9: VTAB 12 to position the cursor in formatted screen output. It is a short assembly language program that uses the Applesoft ampersand (&) command.

USING THE PROGRAM

To use the program, BRUN TAB XY. The general syntax for the utility is *&X,Y* where *X* is an integer number or expression for the HTAB, and *Y* is an integer number or expression for the VTAB. *X* can take on any value from 1-80. (Use the range 1-40 or 1-80 depending on whether you can display 40 or 80 columns with your computer.) *Y* must be in the range 1-24. If you input a value outside these ranges, a beep will sound. After BRUNning TAB XY, you can position the cursor at horizontal position 9 and vertical position 12 simply by using the statement &9,12.

ENTERING THE PROGRAM

Please refer to Appendix A for help in entering the program in **Listing 6**. If you key it in from the Monitor, save it to disk with the command:

BSAVE TABXY,A$3A4,L$2B

HOW IT WORKS

In addition to being a handy ampersand utility, the program provides a good illustration of three Apple ROM subroutines for use in your assembly language programming. GETBYT is a routine that gets an expression (or number), evaluates it, and enters the result in the 6502 X-Register. The expression it handles is the one pointed at by TXTPTR, a vector located on zero page at 184 and 185 ($B8 and $B9). Applesoft constantly updates TXTPTR to the next character in the Applesoft program. In the case of TAB XY, the next expression after the & is the X (HTAB) value.

COMBYTE does essentially the same thing as GETBYT, except that COMBYTE also looks for a comma. If it finds one, it increments TXTPTR and then evaluates the expression. Therefore, the Y (VTAB) expression in TAB XY is obtained, evaluated and stored in the X-Register by COMBYTE.

TABV does the actual work in the TAB XY program. The Monitor routine positions the cursor at the horizontal position (column) given in CH ($24), and at the vertical position (line) given in the Accumulator.

LISTING 6: TABXY

```
                        ;
                        ;                 TABXY
                        ;
                        ;          by S. Scott Zimmerman
                        ;
                        ;           Copyright (c) 1983
                        ;           by MicroSPARC Inc.
                        ;            Concord, MA 01742
                        ;           All Rights Reserved
                        ;
                        ;
                        ;  After BRUNning TABXY to set &-vector,
                        ;  use the syntax in Applesoft:   &X,Y
                        ;  where X is the HTAB value or expression
                        ;  in the range 1 to 80, and where Y is
                        ;  the VTAB value in the range 1 to 24.
                        ;
                        ;
                                 ORG $3A4          ;Top of page 3
                        ;
                        ; Apple Monitor addresses and routines:
                        ;
                        CH       EQU $24           ;Monitor HTAB value
                        AMPER    EQU $3F5          ;Ampersand vector
                        GETBYT   EQU $E6F8       ;Mon rtn: Evaluate expression
                        COMBYTE  EQU $E74C  ;Mon: Check for ","; eval exprsn
                        TABV     EQU $FB5B         ;Monitor tab routine
                        BELL     EQU $FF3A         ;Monitor beep routine
                        ;
                        ;  Initialize the &-vector:
                        ;
03A4  A9 4C                      LDA #$4C          ;Opcode for JMP
03A6  8D F5 03                   STA AMPER         ;Put at &-vector adrs
03A9  A9 B4                      LDA #TABXY        ;Get LOB of starting adrs
03AB  8D F6 03                   STA AMPER+1       ;Stuff it for the JMP
03AE  A9 03                      LDA #TABXY/       ;Get HOB of starting adrs
03B0  8D F7 03                   STA AMPER+2       ; and stuff it also
03B3  60                         RTS
                        ;
                        ;  The main program starts here:
                        ;
03B4  20 F8 E6          TABXY    JSR GETBYT        ;Evaluate X (HTAB) value
03B7  CA                         DEX               ;Adjust range 0 - 79
03B8  86 24                      STX CH            ;Save X in monitor CH
03BA  20 4C E7                   JSR COMBYTE       ;Evaluate Y (VTAB) value
03BD  CA                         DEX               ;Adjust range 0 - 23
03BE  E0 18                      CPX #24           ;Is Y greater than 23?
03C0  B0 0A                      BCS ERROR         ;Yes, it's an error
03C2  8A                         TXA               ;Put Y value in accum.
03C3  A6 24                      LDX CH            ;Check the X value
03C5  E0 50                      CPX #80           ;Is X greater than 79?
03C7  B0 03                      BCS ERROR         ;Yes, it's an error
03C9  4C 5B FB                   JMP TABV          ;Go to monitor routine
                        ;
                        ;  Beep if there's an error:
```

```
    03CC  4C 3A FF  ERROR    JMP BELL          ;Sound the alarm
                             ;

000  ERRORS

03A4  HEX START OF OBJECT
03CE  HEX END OF OBJECT
002B  HEX LENGTH OF OBJECT
95C1  HEX END OF SYMBOLS
```

Verify and Lock

Verify and Lock modifies the DOS VERIFY command to prevent you from accidentally deleting or writing over files you meant to keep.

by Doug Denby

Have you ever deleted a file by mistake? There are many utilities on the market to recover deleted files — that wasn't my problem. Instead, in the process of developing a program, I would sometimes write over previous (and sometimes better) versions of a program by using the SAVE command.

Adopting the habit of appending version numbers to the file names (e.g., TEST1, TEST2, TEST3, etc.) partially solved my problem, but it wasn't a complete solution. Often I couldn't remember the most recent version number (and I am too lazy to catalog the disk before each SAVE). I tried to develop the habit of locking each version as I saved it, but my lack of resolve got the better of me. Finally, I decided to let DOS do it for me.

BUILT-IN LOCKING

After some feeble attempts at modifying the SAVE command, I came up with a better method — modifying the VERIFY command. Why use VERIFY instead of SAVE? Changing the SAVE command would protect only my Applesoft programs. I do some machine language programming and memory (picture) storage and wanted to protect all file types with a single patch.

My objective was to make the VERIFY command exit through the LOCK command whenever it was executed. This would automatically lock every file that is SAVEd or BSAVEd.

When I discovered that the VERIFY code occupied only four bytes, I wondered how I could alter that type of compressed code. I could only replace the four bytes with a three-byte instruction to jump elsewhere to perform the double command I wanted.

FINDING SPACE FOR THE PATCH

The replacement command (VERIFY and LOCK) would have to first load the Accumulator with the File Manager's opcode for VERIFY, then jump to the File Manager subroutine, and finally jump to the LOCK command (a total of eight bytes):

```
LDA #$0C
JSR $A277
JMP $A271
```

I could have put the new VERIFY & LOCK command in the empty space at $BA59. But since I had already put a few other DOS patches there, I needed a new open memory area in DOS.

THE ERROR MESSAGE TABLE

The Error Message Table is a sequential file of phrases imbedded in DOS. Many of the messages are longer than needed, so I decided to shorten one of them to free up some memory in the table. The end of each phrase is marked by BIT7 of the final byte of the phrase. All other bytes have BIT7 in the off (0) condition except the last byte of the phrase. A separate table keeps track of the offset locations in the message table that point to the start of each phrase. Therefore, shortening a phrase is easy. All I needed to do is turn on BIT7 of the last character of the shortened message.

I changed the NO BUFFERS AVAILABLE to NO BUFFERS. The meaning remains clear and 10 extra bytes are available.

MAKING THE PATCH

First, Enter the Monitor by typing Call -151. You should see the * prompt. Now alter the original VERIFY command to point to the new location of the VERIFY & LOCK command by typing:

```
A27D:4C FF A9
```

Now enter the new command with the line:

```
A9FE:D3 A9 0C 20 77 A2 4C 71 A2
```

or use the following Applesoft routine:

```
10  REM  VERIFY AND LOCK VIA LAM
20 A$ = "A27D:4C FF A9 N A9FE:D3 A9 0C 20 77 A2 4C 71 A2 N
    D9C6G"
30  FOR I = 1 TO  LEN (A$)
40  POKE 511 + I, ASC ( MID$ (A$,I)) + 128
50  NEXT
60  CALL  - 144
```

Your DOS 3.3 is now altered. To put the alteration permanently on a disk, just INIT a new disk. From then on, every time that disk is booted, the modified DOS will be placed in the machine.

All DOS commands act as before except that every SAVE, BSAVE and VERIFY command also locks the file if it is verifiable by DOS 3.3. This patch does not affect TEXT file commands in any way.

Apple IIe Cast of Characters

This demonstration program displays the four character sets available on the Apple IIe, IIc and IIGS

by Sandy Mossberg

APPLE II AND IIe CHARACTERS

For the programmer, character generation is the most primal function of a computer. An eight-bit character set must contain exactly 256 ($100) symbols, each represented by a single numeric code. On page 15 of the *Apple II Reference Manual* there is a table that lists the single character set of the Apple II Plus. By using the standard character output routine (COUT, $FDED), ASCII codes 0-63 ($00-$3F), 64-127 ($40-7F) and 160-255 ($A0-$FF) represent inverse, flashing and normal visible characters, respectively.

Codes 128-159 ($80-$9F) are invisible control characters, many of which initiate a hardware or firmware action, e.g., <CTRL>D (134, $84) tickles DOS, <CTRL>L (140, $8C) produces a formfeed and <CTRL>M (141, $8D) forces a carriage return.

With the Apple IIe's near mandatory 80-column card in the auxiliary slot (which simulates slot 3), no longer is simplicity the keyword. With the card inactive, two separate character sets (primary and alternate) may be utilized. With the card active, two different sets (primary and alternate) become available. That makes four distinct character sets built into the Apple IIe package. You may be conjuring up images of Greek or Hebrew alphabets, but all characters are "American" and one set differs from another because the meaning of certain numeric codes changes. Are you still with me?

IIe SOFT SWITCHES

Before we talk about the character sets, we need to discuss soft switches. The IIe contains numerous soft switches that control and detect various states of the machine. Each switch uses three memory locations — one for turning the switch on, one for turning it off and one for reading the on/off condition. The display soft switches are listed on page 28 of the *Apple IIe Reference Manual.*

THE DEMONSTRATION PROGRAM

Listing 7 is an Applesoft program that enables you to display the four character sets. If you run the program with the card inactive, you may toggle between the primary and alternate sets. The same is true with the card active in 40- or 80-column mode.

To permit the Apple to generate each character set, the Applesoft PRINT statement is bypassed and COUT is used directly.

HOW IT WORKS

Lines 190-230 define the machine language subroutines that are poked into page 3 by **lines 240-250**. Locations are given in decimal and the labels correspond to the equates in **line 260**.

Lines 270-330 list the soft switches to be used and their equates. To activate or deactivate most switches, the appropriate location must be written to, i.e., a value must be POKEd into it. To determine the on/off state, the designated location must be read, i.e., a valued must be PEEKed from it. A number larger than 127 ($7F) indicates that the switch is on; if less than 128 ($80), the switch is off.

Line 350 starts the ball rolling. After homing the cursor and clearing the screen, ALTCHARSET is read. If the switch is on, **Line 710** prints ALTERNATE CHARACTER SET on the top line and control passes to **line 380.**

If ALTCHARSET is off, **line 360** calls the subroutine that outputs PRIMARY CHARACTER SET. **Line 380** prints the ones place column headers by poking numbers from

0-15 into location $07 (NYBBLE) and printing the row of hex numbers using PRHEX (Monitor ROM) and OUTSP (Applesoft ROM).

Line 400 starts the FOR-NEXT I-loop by printing the tens place row headers. When the card is active, ASCII codes 0-31 ($00-$1F) and 128-159 ($80-$9F) represent control characters. With the card inactive, only negative ASCII (high bit set) control characters are present.

Since control characters are invisible and may evoke an unwanted action, they must be filtered out and displayed in another manner. **Line 440** tests the status of the 80-column card (regardless of whether it is in 40- or 80-column mode) by checking if slot 3 has control of the output hook directly or via DOS. Testing the input hook would be equally valid. Finding $C3 (195) in the high-order byte of either location confirms an active card.

Do not make the mistake of using the 80COL switch to test for card activity, since it would be off if the active card were in 40-column mode (and the program would not work if run from this mode).

Line 460 checks for all possible control codes. If none is found, flow is routed to **lines 530-540** where the J-loop pokes the ASCII code into location $06 (CHAR) and uses COUT to print each successive row of 16 ($10) characters.

If a control character is encountered with the 80-column card active, **lines 490 and 520** print an arrow followed by a row of normal uppercase characters. In this way, control characters are designated by an arrow following the row header. With an inactive card, positive ASCII control characters do not exist and **line 490** is bypassed.

Switching Sets

When the character set is completed, **line 560** prints the message "SWITCH or END (ANY KEY/E)?" and **line 570** gets the input. If E is pressed, **line 580** terminates the program. Any other keypress switches character sets. The simplest way to do this would be to toggle the display by reading ALTCHARSET. I chose a more circuitous routine to illustrate several points.

If the card is inactive (tested in **line 610**), text page 1 contains a single set of locations extending from 1024-2047 ($400-$7FF), and screen characters may be read (peeked) directly by accessing the desired location. **Line 640** does just that by reading the first character on the screen, converting it to positive ASCII (screen characters are negative ASCII), and comparing it to A. Finding this letter indicates that the alternate character set is on the screen and that the primary set must be toggled in. If A is not the first character on the screen, **line 650** restores the alternate set.

Reading the screen with the 80-column card active is more complicated. Having 80 columns means that twice as may locations are required on text page 1. The IIe handles this situation by assigning odd locations, i.e., 1, 3, 5, etc., to main memory (PAGE2 off) and even locations, i.e., 2, 4, 6, etc., to auxiliary memory (PAGE2 on) which is contained on 1K of card ROM.

Thus, to read the first location (zero) on the first line (1024,$400), PAGE2 first must be turned on (**line 690**). If you wanted to read the second location on the first line, 1024 would still be PEEKed but PAGE2 would be off. Be sure to turn PAGE2 off again before executing another command (**line 640**). At first, this process appears a bit unwieldy, but as you gain experience, it is downright cumbersome. The theory and an 80-column display map are found on pages 30-32 of the new reference manual.

CHARACTER SET DIFFERENCES

After running the demo program several times with the card in either state, the subtle differences between each of the four character sets will surface. **Table 1** summarizes the features of each set. Unfortunately, Table 2-6 (page 20) in the new reference manual is inaccurate (codes $80-$9F always are control characters and do not print uppercase letters), incomplete (with the card active, codes $00-$1F also are control values), and misleading ("upper-case letters" and "lower-case letters" include several special character, and "special characters" include numerals). **Table 1** may be more meaningful to you.

The primary character set is standard with the card inactive, whereas the alternate set is standard with an active card. The Apple IIe 80-Column Text Card Manual states authoritatively on page 31 that the FLASH command "is not available while the card is active." Since the primary set does appear to contain flashing "upper-case" characters with the card active, let us challenge this assertion. Enter the following code:

```
10 POKE 49166,0: FLASH: PRINT "IMPORTANT": NORMAL
```

Type PR#3 to activate the card, run the program, and see how well the FLASH command functions. We simply have enabled the primary character set by writing to location 49166. From the immediate mode type POKE 49167,0 to restore the alternate set, and see that the flashing characters are replaced by inverse ones. Run the program when the cursor is on the bottom line and note the line of inverse spaces below the flashing message. Scrolling causes this undesired effect. It would be wiser, therefore, to restrict a flashing message to a screen location that does not cause scrolling. In practice, a more elegant method of flashing 80-column characters is to do it temporarily. Consider this second one-liner:

```
10 HOME: VTAB 10: HTAB 1: INVERSE: PRINT "IMPORTANT": NORMAL:
     POKE 49166,0: FOR I=1 TO 1000: NEXT: POKE 49167,0
```

The "IMPORTANT" message is printed inversely and immediately converted to flashing by writing to location 49166. A short timing loop is executed. Restoring the alternate character set stops the flashing and you end up with a neat method of attracting attention. Any other inverse "upper-case" characters on the screen also will flash momentarily, and inverse "lower-case" characters temporarily will be converted to "special" characters.

TABLE 1: Character Sets of the Apple IIe

	40-Column		80-Column	
	Primary	**Alternate**	**Primary**	**Alternate**
$00-$1F	Uppercase Inverse	Uppercase Inverse	Control	Control
$20-$3F	Special Inverse	Special Inverse	Special Inverse	Special Inverse
$40-$5F	Uppercase Flash	Uppercase Inverse	Uppercase Flash	Uppercase Inverse
$60-$7F	Special Flash	Lowercase Inverse	Special Flash	Lowercase Inverse
$80-$9F	Control	Control	Control	Control
$A0-$BF	Special Normal	Special Normal	Special Normal	Special Normal
$C0-$DF	Uppercase Normal	Uppercase Normal	Uppercase Normal	Uppercase Normal
$E0-$FF	Lowercase Normal	Lowercase Normal	Lowercase Normal	Lowercase Normal

Special: S P ! " # $ % & ' ( ) * + , — . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
Uppercase: @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\ ] ^ _
Lowercase: a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~ DEL

LISTING 7: CHARACTER SET DEMO

```
100   REM *****************************************************
110   REM *                                                   *
120   REM * Apple IIe CHARACTER SET DEMO:Displays alternate and *
130   REM *primary character sets with 80 column card on and off*
140   REM *                                                   *
150   REM *                by Sandy Mossberg                  *
160   REM *     Copyright (C) 1983 by MicroSPARC, Inc.        *
170   REM *****************************************************
180   REM    Machine language subroutines and equates:
190   REM    768  COUTSP LDA $06
200   REM    770         JSR $FDED   ;print char. in $06 (ASCII)
210   REM    773  OUTSP  JMP $DB57   ;print space
220   REM    776  PRHEX  LDA $07
230   REM    778         JMP $FDE3   ;print low nybble in $07
240   FOR I = 768 TO 780: READ N: POKE I,N: NEXT I
250   DATA 165,6,32,237,253,76,87,219,165,7,76,227,253
260 CHAR = 6:NYBBLE = 7:COUTSP = 768:OUTSP = 773:PRHEX = 776
270   REM    Soft switch equates:
```

```
280  REM       ALTCHARSET: OFF  (FALT) = 49166 ($C00E)
290  REM                   ON   (NALT) = 49167 ($C00F)
300  REM                   READ (RALT) = 49182 ($C01E)
310  REM       PAGE2:      OFF  (FPG2) = 49236 ($C054)
320  REM                   ON   (NPG2) = 49237 ($C055)
330 FALT = 49166:NALT = 49167:RALT = 49182:FPG2 = 49236:NPG2 =
     49237: REM
340  REM    Read character set and print it on top line.
350  HOME : IF  PEEK (RALT) > 128 THEN  GOSUB 710: GOTO 380
360  GOSUB 720: REM
370  REM    Print column headers ("ones place").
380  PRINT : HTAB 5: FOR I = 0 TO 15: POKE NYBBLE,I: CALL PRHEX:
     CALL OUTSP: NEXT I: PRINT : PRINT : REM
390  REM    Print row headers ("tens place").
400  FOR I = 0 TO 255 STEP 16: POKE NYBBLE,I / 16: CALL PRHEX:
     POKE NYBBLE,0: CALL PRHEX: REM
410  REM Determine 80 column card (slot 3) activity by checking
420  REM    for $C3 (195) in high order byte of output hook
     (55,$37)
430  REM    or true output handler of DOS (43604,$AA54).
440  IF  PEEK (55) <  > 195 AND  PEEK (43604) <  > 195 THEN 520
450  REM    80 column card active. Non-control characters print
     normally.
460  IF I <  > 0 AND I <  > 16 AND I <  > 128 AND I <  > 144
     THEN 530: REM
470  REM  80 column card active. Convert positive ASCII control
480  REM    characters to normal characters and mark them.
490  IF I = 0 OR I = 16 THEN  PRINT "->";: FOR J = I TO I + 15:
     POKE CHAR,J + 192: GOTO 540: REM
500  REM 80 column card active or inactive. Convert neg. ASCII
510  REM control characters to normal characters and mark them.
520  IF I = 128 OR I = 144 THEN  PRINT "->";: FOR J = I TO I +
     15: POKE CHAR,J + 64: GOTO 540
530  PRINT "  ";: FOR J = I TO I + 15: POKE CHAR,J
540  CALL COUTSP: NEXT J: PRINT : NEXT I: REM
550  REM    Control line - switch character sets or end.
560  VTAB 23: PRINT "SWITCH or END (ANY KEY/E)? ";
570  VTAB 23: HTAB 28: GET A$: REM
580  REM    End it all.
590  IF A$ = "E" THEN  END : REM
600  REM    Switch. Read 80 column card activity.
610  IF  PEEK (55) = 195 OR  PEEK (43604) = 195 THEN 690: REM
620  REM Read first screen character on text page 1. If "A" then
630  REM switch to primary set; if "P", switch to secondary set.
640  IF  CHR$ ( PEEK (1024) - 128) = "A" THEN  POKE FALT,0: POKE
     FPG2,0: GOSUB 720: GOTO 570
650  POKE FPG2,0: POKE NALT,0: GOSUB 710: GOTO 570: REM
660  REM  If 80 column card active, to read first char. on text
670  REM    page 1 (even numbered column), turn auxiliary memory
680  REM    ("PAGE2") on. Truth is stranger than fiction!
690  POKE NPG2,0: GOTO 640: REM
700  REM    Subroutines to print title on top line.
710  PRINT : VTAB 1: PRINT "ALTERNATE CHARACTER SET:": RETURN
720  PRINT : VTAB 1: PRINT "PRIMARY CHARACTER SET:  ": RETURN
```

Applesoft Tricks

Six short tricks provide solutions to common Applesoft programming problems without resorting to machine language routines.

by Craig Peterson

As powerful as Applesoft BASIC is, once in a while there are things that you would like it to do that it doesn't. One example is permitting an INPUT command that allows commas as part of a person's name (without throwing part of your input away and giving you and EXTRA IGNORED message).

Many times the solutions to dilemmas like these are found in machine language programs. But many of you are unfamiliar with machine language, and often these routines conflict with one another because they occupy the same area of memory. Hex page $300 of the Apple's memory has been used this way so much that I'm surprised the memory chips don't wear out. It would be much simpler to program some of these routines in Applesoft and that's what I've done. A word of caution: If you plan to use a compiler on your programs, some of these methods may not work. In particular, those that use locations 131 and 132 and those that revise the Applesoft pointers.

INPUT ANYTHING TRICK

One of the most common difficulties in Applesoft is allowing a user input that includes commas, colons, quotation marks and the like. The standard INPUT statement won't permit them. Instead, you get just that part of the input up to the first special character and then an EXTRA IGNORED message. Using some of the above characters in your input, try typing in and running the following lines:

```
100 PRINT "?";: GOSUB 130
110 PRINT IN$
120 END
130 CALL 54572: FOR B = 512 TO 768: IF PEEK (B) <> 0 THEN NEXT
140 IN$="":AD = VAL (IN$) + PEEK (131) + 256 * PEEK (132): POKE
     AD,B - 512: POKE AD + 1,0: POKE AD + 2,2:IN$ = MID$ (IN$,
     1): B = 768: NEXT : RETURN
```

Lines 130 and 140 comprise an input subroutine that will gather all the characters you type in (except for the Apple's <ESC> key cursor moves) and place them into IN$. Unlike GET statement input routines you may have seen before, this subroutine will create absolutely no string garbage while doing its work. It can be used for disk input as well as keyboard input. And it's almost as fast as the standard INPUT command because it actually uses part of Applesoft to gather the input.

How INPUT Works

The two secrets to the operation of this subroutine are the CALL statement and memory locations 131 and 132. Location 54572 is ROM Applesoft's machine language subroutine that accepts the keyboard or DOS input and puts it in the input buffer. A maximum of 239 characters are allowed, just like the normal Applesoft INPUT command.

After the FOR-NEXT loop determines the length of the input, **line 140** finds the Applesoft pointers to the variable IN$, and stuffs the input line length and the input buffer address into these pointers. This is where memory locations 131 and 132 come in. If manipulated properly, (as in the two statements in **line 140**) they will contain the address of the string variable's pointers in memory.

After the input buffer pointers are placed into IN$, the string is relocated into main memory with the MID$ statement. Finally, the dangling FOR-NEXT loop is completed by setting the index B to its finishing value so the NEXT will be satisfied.

THE DO-WHILE TRICK

One criticism directed at the BASIC language is that it does not allow such structured programming statements as DO-WHILE. Instead, BASIC program lines often use a lot of GOTOs to accomplish the same thing, and in long programs these GOTOs can be quite slow in execution. Actually, Applesoft does have a type of DO-WHILE as shown in the following program:

```
200 J = 0
210 FOR I = 0 TO 1:J = J +1:I = A(J) = B: NEXT
```

How DO-WHILE Works

The above example uses a capability of Applesoft called Boolean (or logical) algebra. You use it all the time when you program IF statements. Essentially, it is the mathematics of true and false.

If an expression evaluates as true, then it is given a value of 1. If the expression evaluates as false, it is given a value of 0. The results of such expressions, which can be as complicated as you require, can be placed into a numeric variable. For example, if A$ is the same as B$, the statement I = A$ = B$ places a 1 into I. If they are not equal, a 0 is placed into I. Any valid relational expression can be placed on the right-hand side of the first equal sign, and the I is given a value according to the truth of the statement.

In **line 210** above, the array A(J) is searched for the value in B, which is known to exist somewhere in the array. As long as A(J) is not equal to B, the expression will evaluate to be 0, which will be placed into I. Since I will not have reached its terminating value of 1, the NEXT will continue the loop.

As soon as A(J) = B, this expression will evaluate to be a 1 which will be placed into I. The NEXT will then see that I has reached its final value and will drop out of the loop.

THE ARRAY CLEAR TRICK

Applesoft BASIC allows something that many non-BASIC languages do not — the dynamic allocation of arrays. The capability to dimension arrays to a variable number of elements right in the middle of running programs can give certain programs a great deal of flexibility. But once the array has been dimensioned, it cannot normally be redimensioned. If it were possible to delete an existing array, Applesoft would then allow you to redimension that array to a different number of elements. By manipulating one of Applesoft's pointers, it is possible to do this.

The simplest approach to deleting arrays is to delete all the arrays. If you have only one array in your program, or you can afford to delete all of the current arrays, then the following line will accomplish this:

```
300 POKE 109, PEEK (107): POKE 110, PEEK (108)
```

How Array Clear Works

Memory locations 107 and 108 are used by Applesoft as a pointer to the beginning of the array storage. Memory locations 109 and 110 are used as a pointer to the top end of variable storage. By placing 107,108 into 109,110, the array storage is totally deleted, allowing you to redimension any and all arrays without a peep from Applesoft.

It is usually a good idea to immediately follow the above statement with an X = FRE(0) statement to force Applesoft to do its string house cleaning. Since all the arrays have been deleted, the string garbage will normally be swept clean in the blink of an eye. Incidentally, this technique is also a very quick method of reinitializing all the elements of an array to 0 and it is much faster than a FOR-NEXT loop. Just delete the array and then redimension it to the same size.

SELECTIVE CLEARING

If you want to delete only one or two arrays use the technique shown in the following lines:

```
310 DIM A1(100),A2(200),A3(300)
320 AD = 0 * A2(0) + PEEK 131) + 256 * PEEK (132) -7: POKE
     110,AD/256: POKE 109, AD - 256 * PEEK (110)
```

Line 320 will delete array A2(*) and all arrays that were dimensioned after it in the program. In the above example that would include array A3(*). Array A1(*) is left intact.

How Selective Clear Works

The minus 7 term in the first statement of **line 320** is a function of the number of dimensions in the A2(*) array. This minus term should be 5 plus 2 times the number of dimensions.

Since A2(*) was a one-dimensional array, this term was 5 + 2 *1 or 7. If the array had been a two-dimensional array (like A2(*,*)) then the term would have been 5 + 2 * 2 or 9.

For a string array, the beginning of **line 320** should be changed as follows:

```
320 AD = 0 * VAL (A2%(0)) + . . .
```

By dimensioning last those arrays you wish to delete, you can selectively redimension them according to the needs of your program.

Selective deletion of arrays is accomplished in the same way as the total array deletion, except that instead of placing the beginning array storage into the end-of-variables pointer, we place the beginning of the specified array there. Therefore, that array and all arrays stored above it are deleted. Our old friends, memory locations 131 and 132 help us find AD, the address of the beginning of the array.

THE LOWERCASE TRICK

Applesoft has a built-in talent for lowercase letters. Assuming you have a printer plugged into slot 1, try running the following line:

```
400 PRINT CHR$(4);"PR#1": PRINT: PRINT "L";:POKE 243,32: PRINT
    "OWER ";: NORMAL: PRINT "C";: POKE 243,32: PRINT "ASE":
    NORMAL: PRINT CHR$(4);"PR#0"
```

How Lowercase Works

Memory location 243 is a special flag that tells Applesoft if lowercase conversion is to be performed on the output of alphabetical and a few other special characters. POKEing a 32 into this location causes subsequent characters to be converted from uppercase to lowercase. POKEing a 0 into this location or using the NORMAL command cancels the conversion.

Depending on the printer card you use, you may have noticed some strange characters printed onto your screen. This would happen if your Apple doesn't have a lowercase character set in its character generator. If you have installed a lowercase chip in your Apple, or have an Apple IIe, IIc or IIGS, you'll see lowercase characters printed on the screen. Although it is somewhat involved to print a mixture of uppercase and lowercase characters, there are situations in which the technique shown in **line 400** above can be simpler than other methods for achieving lowercase output.

THE ON-GOTO TRICK

IF-THEN and IF-THEN GOTO statements are probably the most familiar methods used to conditionally change program flow by jumping to new line numbers. When used in this way, the IF statement usually becomes the only statement on the line because if the condition is satisfied, the jump is taken; if it's not satisfied, the rest of the line is ignored. Another statement that can provide the same flow control and also give you greater flexibility is the ON-GOTO statement. Here's an example:

```
500 ON A = B + 1 GOTO 800: PRINT MSG$: ON A >= C GOTO 850: PRINT
    M2$
```

How ON-GOTO Works

ON-GOTO usually selects from a list of line numbers to which to jump based on the expression that follows the ON keyword. You may also place a relational expression after the ON. This expression will evaluate to either a one or zero, depending on whether the expression is true or false.

If the expression is true, the one will select the first (and only) line number to which to jump. If it's false, the zero will cause the program flow to fall through to the next statement on the same line. In this way, a program flow decision can be made on a line yet you can still use the remainder of the line.

THE NUMBERS FILE TRICK

A disappointing feature of DOS 3. is the speed at which text file information can be written to or read from a disk. Something as simple as saving a large array of numbers can be time-consuming. Machine language subroutines are available to help alleviate the problem by converting text to binary numbers. But it's much nicer to do it with a somewhat standard Applesoft/DOS statement:

```
600 DIM ARRAY(7,8)
610 AD = 0 * ARRAY(0,0) + PEEK (131) + 256 * PEEK (132): PRINT
    D$;"BSAVE ARRAY,A";AD;",L"; 5 * 8 * 9
620 AD = 0 * ARRAY(0,0) + PEEK (131) + 256 * PEEK (132): PRINT
    D$;"BLOAD ARRAY, A";AD
```

The setup of **lines 610 and 620** is very similar to that used in previous examples. You probably recognize 131,132 locating the beginning of the array variables. However, let me explain the 5 * 8 * 9 term in **line 610**. The five in this product is used because the array is a floating point array whose numbers take up five bytes of storage space in memory. If this were an integer array (%), this number would be a two instead of a five, because integer array numbers take only two bytes of memory.

The eight and the nine represent the number of elements in each dimension of the array, which, because of the zeroth element, is always one more than the dimension numbers in the DIM statement.

To use **line 620**, the array must have been dimensioned to the same size as the one that is being BLOADed from the disk. Otherwise, the data overflows your dimensioned array space.

Note that this method cannot be used to store and retrieve string arrays from the disk. It only works with floating point or integer numeric arrays.

How Numbers File Works

In **line 610** above, the numeric data is saved as a binary file instead of a text file because binary files are saved and loaded much faster than text files. The actual section of memory in which the array numbers are stored is saved onto the disk.

Text file PRINT and INPUT statements are extremely slow to execute. You should find that the method used in **lines 610 and 620** is four to five faster on the average than using a normally written or read text file.

An additional advantage of using the above technique is that data usually takes less storage space on the disk. A typical decimal fraction expressed in scientific notation can take as many as 15 bytes of disk storage. If you used the above method, this floating point number would take only five bytes. You can save about 50% in disk storage space, depending on the nature of the numbers in the array.

DOS Catalog Dater

Make the date your file was last updated part of your program's file name with this easy modification to DOS.

by Art Mena

One of the most convenient features of the Apple II Pascal operating system is that is records the date on which a disk file was written. The date is a part of that file's directory entry and enables you to keep track of when files were last updated. Large timesharing computer systems also perform this function, although for archiving purposes rather than for programmer convenience.

It is possible to modify DOS to automatically add the date as part of the file name. But to understand how to do this, we need to explore how DOS saves a file to a disk.

DOS FORMATTED DISKS

DOS 3.3 tracks are numbered from $0-$22 (0-34). Tracks $0-$2 are normally reserved for DOS. Track $11 (17) contains the Volume Table of Contents (VTOC) and the directory which is where the file names are stored. When you do a CATALOG, DOS reads the directory from track $11 and prints out information pertaining to the active (not deleted) files on the disk. The VTOC occupies track $11 sector $0 ($11/$0). The directory occupies the rest of the track — sectors $1-$F.

Figure 2 shows the format of the directory sectors. Each directory sector can hold 7 directory entries for a maximum of 105 files per disk.

FIGURE 2: *Directory Sector Format*

Byte	Description
$0	Not used
1	Track where the next directory sector is found ($11)
2	Sector where the next directory sector is found
3-A	Not used
B-2D	Directory entry for File 1
2E-50	Directory entry for File 2
51-73	Directory entry for File 3
74-96	Directory entry for File 4
97-B9	Directory entry for File 5
BA-DC	Directory entry for File 6
DD-FF	Directory entry for File 7

The format of each of the directory entries is shown in **Figure 3**. The file name is contained in $3-$20, for a maximum of $1E (30) characters in the file name.

FIGURE 3: *Directory Entry Format*

Relative Byte	Description
$0	Track number of the files track/sector list
1	Sector number of the files track/sector list
2	File type
3-20	File name
21-22	Numbers of sectors occupied by the file

However, if the last character in a file name is not a blank, certain commercial DOS utility programs will treat it as a deleted file, and may cause the file to disappear. Thus we actually have only 29 characters to use for the file name. If we add the date in the format MM/DD/YY then we have 21 characters left for the file name.

CATALOG DATER PROGRAM

The scheme for adding the date to the file name is quite simple. When you SAVE, BSAVE or OPEN a file, DOS searches the directory for the file name you specify by reading each directory sector, starting with $F, into a buffer located at $B4BB DOS first searches sector $F for the file name. If it doesn't find the name, DOS reads in sector $E, then $D, $C and so on. DOS searches until it finds the file name or it reaches the end of the directory.

If DOS finds the file name, it saves an index pointing to the file's directory entry. At this point, my program jumps to a subroutine I call CATDATE and copies the date from location DATE ($BCDF) to the last eight characters of the file name in the directory sector buffer. Then it calls a routine to write the directory sector back out to the disk.

When DOS does not find the file name in the directory, it copies the file name following the SAVE, BSAVE or OPEN command to the first available directory entry. At that point the program jumps to the routine to copy the date to the file name and write the directory sector to the disk.

A search of DOS reveals two locations where we must insert a JSR CATDATE command. The first is $B206 where DOS successfully finds the file name in the directory. The second location, $B22B, is for the case where DOS does not find the file name.

Since I don't think that changes to DOS should be permanent, I put the Catalog Dater routine in place of the INIT function. This way, you cannot initialize a new disk with the Catalog Dater routine incorporated into DOS. The routine will be poked into memory immediately after you boot up DOS using the Hello program. Another reason is that DOS is fairly compact, and there are few empty spaces in which to insert a new subroutine. By disabling INIT, there is plenty of space for CATDATE and other routines.

HOW IT WORKS

The first thing the catalog date program (**Listing 8**) does is check for a SAVE, BSAVE or OPEN command. Since these are the only DOS commands that will alter the contents of a file, the file date will be changed only if the file itself has been changed.

If DOS is performing one of these commands, the program next checks to see if the file is locked by checking bit seven of the file type (see **Figure 3**). It bit seven is one (i.e., the file type is minus), the file is locked and the program does not change the date. If the file is not locked, the date is transferred from location DATE ($BCDF) to the last nine characters of the file name which is located in the directory sector buffer.

This sector is then written to the disk, and the program is finished. Note that the date will be copied to the file name for an OPEN command so that a file's date will be updated for both OPEN, WRITE and OPEN,READ commands. I did this for simplicity, since DOS does not search the directory for a READ or WRITE command. You may decide to add a test for a READ command.

CATALOG DATE HELLO PROGRAM

The routine CATDATE is poked into memory using a modified Applesoft Hello program. The program to determine the date and poke it into memory starting at location DATE ($BCDF, 48351) is shown in **Listings 9 and 10**.

There are two versions. The first is for those who do not have a real-time clock/calendar card installed. The routine will prompt you to type in the correct date, check it for validity and poke it into memory.

The second version reads the month and day from a Mountain Computer Appleclock in slot 4. The year is added in **line 580** and the date is poked into memory. If you have another clock card, you can easily modify this program to read the date from it.

Don't worry about the lowercase characters, just type them in in uppercase. Once CATDATE is poked into memory, it will remain there as long as DOS is not altered or until the Apple is turned off. So if your programming extends past midnight, you must either reboot or rerun Hello to change the date.

21-CHARACTER NAMES

There is one additional change to DOS that you must make in order to use CATDATE. Since the file names are now 21 characters long, you must tell DOS to limit its search to 21 characters. This is done by poking $15 (21) into two locations: $B203 (45608) and $B228 (45571).

However, this brings up a potential problem. If you incorporate CATDATE into DOS and then save the new Hello program to disk, it will have the date as part of the file name. Now, the next time you boot the disk, the unmodified DOS will be loaded into memory and will search for the Hello program. Unfortunately, it will not find a file named Hello, because it is searching for a 30-character file name. Our new Hello program file name has the date in characters 22-29. In order to avoid this problem you can do one of two things. You can either save the Hello program using only unmodified DOS (i.e., CATDATE not installed).

Alternately, you can change the default file name length permanently. To do this, change bytes $03 and $28 on track/sector $02/$01, to $15 (21). Use a program that will read and write individual disk sectors, such as Disk Zap (*Nibble* Vol. 4/No. 3), to read, edit and write to the disk sector. After this modification, DOS will always look for 21 characters in the file name and ignore any other characters.

LISTING 8: CATALOG.DATER

```
                 ;
                 ;              CATALOG.DATER
                 ;
                 ;           BY ARTHUR L. MENA
                 ;           COPYRIGHT (C) 1982
                 ;           BY MICROSPARC, INC.
                 ;           CONCORD, MA  01742
                 ;
                 ;
                 ;   This routine replaces the INIT Function
                 ;
                 ;   These JSR's must be added to DOS to enable
                 ;   CATDATE:
                 ;
                 ;   B206:  JSR $AE8F
```

```
                 ;  B22B:  JSR $AE8F
                 ;
                 ;  Also the file name length must be reduced to $15
                 ;
                 ;  B203:15
                 ;  B228:15
                 ;
                 ;  Decimal locations
                 ;
                 ;   Label      HEX       DEC
                 ; ----------------------------
                 ;   START     $AE8E     44686
                 ;   CATDATE   $AE8F     44687
                 ;   DATE      $BCDF     48351
                 ;   END       $AEBC     44732
                 ;
                 ;   Equates
                 ;
                 CMDINDX  EQU $AA5F
                 WRTDIRSC EQU $B037
                 DIRINDX  EQU $B39C
                 TSTRACK  EQU $B4C6
                 FILTYP   EQU $B4C8
                 DATE     EQU $BCDF
                 ;
                          ORG $AE8E
                 ;
                 ;  Kill INIT function
                 ;
AE8E  00         START    DFC $00
                 ;
                 ;  Determine if this is a SAVE, BSAVE, or OPEN
                 ;  command.
                 ;
AE8F  AD 5F AA   CATDATE  LDA CMDINDX
AE92  C9 04               CMP #$04        ;SAVE Command
AE94  F0 0C               BEQ C2
AE96  C9 30               CMP #48         ;BSAVE
AE98  F0 08               BEQ C2
AE9A  C9 1A               CMP #26         ;OPEN
AE9C  F0 04               BEQ C2
AE9E  C9 00               CMP #00         ;OTHER
AEA0  D0 17               BNE C3
                 ;
                 ;  Check if file locked
                 ;
AEA2  AE 9C B3   C2       LDX DIRINDX
AEA5  BD C8 B4            LDA FILTYP,X
AEA8  30 0F               BMI C3          ;Yes
                 ;
                 ;  Transfer the date from the location DATE to the
                 ;  directory sector buffer.
                 ;
                 ;  The date is stored backwards as follows:
                 ;         DATE
                 ;           !
                 ;          0123456789
                 ;          " RY/DD/MM"
```

```
                     ;           " 28/02/80"  for example is 08/20/82
                     ;
AEAA  A0 08                    LDY #8
AEAC  B9 DF BC  C1             LDA DATE,Y
AEAF  9D DE B4                 STA TSTRACK+24,X
AEB2  E8                       INX
AEB3  88                       DEY
AEB4  10 F6                    BPL C1
                     ;
                     ;  Save Directory sector to disk
                     ;
AEB6  20 37 B0                 JSR WRTDIRSC
                     ;
                     ;  Reset X and return
                     ;
AEB9  AE 9C B3  C3             LDX DIRINDX
AEBC  60                       RTS

000  ERRORS

AE8E  HEX START OF OBJECT
AEBC  HEX END OF OBJECT
002F  HEX LENGTH OF OBJECT
95A5  HEX END OF SYMBOLS
```

LISTING 9: CAT.DATE.NOCLOCK

```
10  REM     **********************
11  REM     *  CAT.DATE.NOCLOCK  *
12  REM     * BY ARTHUR L. MENA  *
13  REM     * COPYRIGHT (C) 1983 *
14  REM     * BY MICROSPARC, INC *
15  REM     * CONCORD, MA. 01742 *
16  REM     **********************
120  REM
130  REM
140  REM   This version of the program
150  REM   is for those who do not have
160  REM   a clock/calendar card in
170  REM   their Apple
180  REM
220  REM
230  REM   Poke CAT.DATE into memory
240  REM
250 B$ =  CHR$ (7) +  CHR$ (7) +  CHR$ (7): REM  3 Bells
260  RESTORE
270  FOR I = 44686 TO 44732
280  READ D: POKE I,D
290  NEXT I
300  REM
310  REM   Poke JSR CAT.DATE into memory
320  REM
330  POKE 45611,32: POKE 45612,143: POKE 45613,174
340  POKE 45574,32: POKE 45575,143: POKE 45576,174
```

```
350  REM
360  REM   Change file name length to 21
370  REM
380  POKE 45608,21: POKE 45571,21
390  REM
400  REM
410  DATA 0,173,95,170,201,4,240,12,201,48,240,8,201,26,240,4,
     201,0,208,23,174,156,179,189,200,180,48,15,160,8,185,223,18
     8,157,222,180,232,136,16,246
420  DATA 32,55,176,174,156,179,96
430  REM
440  REM
450  REM   Input date from keyboard
460  REM
470  TEXT : HOME
480  VTAB 5: HTAB 10: PRINT "CAT.DATE INSTALLED": PRINT
490  PRINT " INPUT THE CURRENT DATE": PRINT
500  PRINT
510  REM
520  VTAB 10: INPUT " WHAT IS THE CURRENT MONTH (1-12) ?";MN$
530  IF  VAL (MN$) < 1 OR  VAL (MN$) > 12 THEN  PRINT B$;"MONTH
     INCORRECT": GOTO 520
540  VTAB 12: INPUT " WHAT IS THE CURRENT DAY   (1-31) ?";DA$
550  IF  VAL (DA$) < 1 OR  VAL (DA$) > 32 THEN  PRINT B$;"DAY
     INCORRECT": GOTO 540
560  VTAB 14: INPUT " WHAT IS THE CURRENT YEAR (00-99) ?";YR$
565  IF  VAL (YR$) < 0 OR  VAL (YR$) > 99 THEN  PRINT B$;"YEAR
     INCORRECT": GOTO 560
570  REM
580 DA$ =  STR$ ( VAL (DA$))
590  IF  VAL (DA$) < 10 THEN DA$ = "0" + DA$
600 MN$ =  STR$ ( VAL (MN$))
610  IF  VAL (MN$) < 10 THEN MN$ = "0" + MN$
620  REM
630 DT = 48351: REM  $BCDF
640 DT$ = MN$ + "/" + DA$ + "/" + YR$ + " "
650  PRINT : PRINT
660  PRINT DT$" HAS BEEN INSTALLED AS THE     CURRENT DATE"
670  REM
680  REM  Poke date into memory
690  REM
700 J = 8
710  FOR I = 0 TO  LEN (DT$) - 1
720  POKE DT + J, ASC ( MID$ (DT$,I + 1,1)) + 128
730 J = J - 1
740  NEXT I
750  END
```

LISTING 10: CAT.DATE.CLOCK

```
10  REM     **********************
11  REM     *    CAT.DATE.CLOCK   *
12  REM     * BY ARTHUR L. MENA  *
13  REM     * COPYRIGHT (C) 1983 *
14  REM     * BY MICROSPARC, INC *
15  REM     * CONCORD, MA. 01742 *
16  REM     **********************
120  REM
130  REM  Apple clock version
140  REM
150  REM   This version of the CATALOG
160  REM   DATE program assumes you have
170  REM   a Mountain Computer Appleclock
180  REM   in slot #4.  This program can
190  REM   be easily modified to read the
200  REM   date from other clock cards.
210  REM   Consult the clock manuals for
220  REM   details.
230  REM
270  REM
280  REM   Poke CATDATE into memory
290  REM
300  RESTORE
310  FOR I = 44686 TO 44732
320  READ D: POKE I,D
330  NEXT I
340  TEXT : HOME : PRINT : PRINT "   CAT.DATE INSTALLED"
350  REM
360  REM   Poke JSR CATDATE into memory
370  REM
380  POKE 45611,32: POKE 45612,143: POKE 45613,174
390  POKE 45574,32: POKE 45575,143: POKE 45576,174
400  REM
410  REM   Change file name length to 21
420  REM
430  POKE 45608,21: POKE 45571,21
440  REM
450  DATA 0,173,95,170,201,4,240,12,201,48,240,8,201,26,240,4,
     201,0,208,23,174,156,179,189,200,180,48,15,160,8,185,223,18
     8,157,222,180,232,136,16,246
460  DATA 32,55,176,174,156,179,96
470  REM
480  REM  Read date from Mountain Computer
490  REM  Apple clock in slot four
500  REM
510 D$ =  CHR$ (4)
520  PRINT D$"IN#4"
530  PRINT D$"PR#4"
540  INPUT " ";T$
550  PRINT D$"IN#0"
560  PRINT D$"PR#0"
```

```
570  PRINT
580 YR$ = "/87 "
590 DT$ =  LEFT$ (T$,5) + YR$
600 DT = 48351: REM  $BCDF
610  PRINT : PRINT : PRINT DT$" HAS BEEN INSTALLED AS THE
     CURRENT DATE"
620  REM
630  REM  Poke date into memory
640  REM
650 J = 8
660  FOR I = 0 TO  LEN (DT$) - 1
670  POKE DT + J, ASC ( MID$ (DT$,I + 1,1)) + 128
680 J = J - 1
690  NEXT I
700  END
```

DOS Error Message and Command Changer

This short Applesoft routine lets you modify DOS 3.3 commands ad error messages to personalize your programs.

by Donald Miller, M.D.

Rewritten error messages are one way to personalize your Apple and DOS. Changing error messages actually serves little purpose, but WHAT PROGRAM? or DOS BOOBOO may be easier to swallow than FILE NOT FOUND or SYNTAX ERROR. However, changing commands, besides personalizing, can be used to protect your disks, especially by changing the INIT, CATALOG and SAVE commands. (See Craig Crossman's article in *Nibble* Vol.2/No.3 for an excellent discussion of this.)

USING COMMAND CHANGER

After you see the title page, the program asks if you want to change error messages or DOS Commands. Each standard message or command is then displayed in the order it appears in the current DOS. You are asked to change that message or to go to the next.

If a change is to be made, the new message can be typed in. The new error messages may not be longer than the old ones although they can be shorter and include punctuation and spaces. However, new commands must contain the same number of letters as the old ones, and you cannot use spaces or punctuation. After the change is entered, there's an opportunity to correct it; otherwise the change is POKEd into DOS and temporarily stored in RAM.

The changes only affect DOS commands; for example, the RUN command will only be affected if it is used with a file name.

SAVING THE CHANGES

The program allows you to easily create two binary files by capturing the newly created configurations in RAM. These binary files could then be added to your disks and the Hello program of each could be modified to BLOAD the files. As an alternative, after exiting the program, a new disk could be initialized; this disk will contain your personalized messages without any further modification.

ENTERING THE PROGRAM

To key in the program, type the program as shown in **Listing 11** and save it to disk with the command:

SAVE COMMAND.CHANGER

HOW IT WORKS

Lines 20-50 prompt you to change error messages or DOS commands. **Lines 50 and 90** specify the starting and the ending addresses, as well as the length of the commands and error messages, respectively. **Lines 100-110** PEEK the starting address and each successive address of each message and store them as a concatenated string.

Line 120 checks to see if the character is a negative ASCII character (high bit set), the last character of that message. This line also checks to see if it is the end of all the messages or commands.

Lines 150-190 display the "PEEKed" message and ask for a new entry. **Line 200** sets up an inverse field and limits the number of characters to the length of the PEEKED message. **Line 210** positions the cursor.

Line 220 GETs each new character and creates another concatenated string. This method allows punctuation to be used. If you press the <RETURN> key at this point, the display may not look very good, but your error messages can be put on several lines instead of all on one.

Lines 230-240 allow corrections. **Line 250** calculates the starting address, minus one, of the current message. **Lines 260-270** use MID$ to extract each character and in **line 280** the ASCII code of each is POKEd into memory.

Line 300 makes the last character negative by adding 128 (setting the high bit) to the ASCII code. **Lines 320-380** print the options to save the new messages. A binary file to capture the changes in RAM is created in **line 400. Lines 440-500** are the title page.

TABLE 2: Addresses (48K)

	Commands	Error Messages
Decimal	43140-43271	43380-43581
Hexadecimal	\$A884-\$A907	\$A974-\$AA3D

LISTING 11: COMMAND.CHANGER

```
10   REM  COMMAND.CHANGER
12   REM  BY DONALD MILLER
13   REM  COPYRIGHT (C) 1983
14   REM  BY MICROSPARC INC
15   REM  CONCORD, MA 01742
20   GOSUB 440: HOME : VTAB 12: PRINT "CHANGE ERROR CODES OR
      COMMANDS ? (E/C) ";: GET N$
30   IF N$ = "E" THEN 90
40   IF N$ <  > "C" THEN 20
50   IF N$ = "C" THEN AD = 43140:B = 43272:C = 133: PRINT N$:
      VTAB 16: HTAB 1: INVERSE : PRINT "NO NUMBERS, PUNCTUATION
      MARKS OR SPACES MAY BE USED WHILE CHANGING COMMANDS"
60 A = AD
70   NORMAL
80   FOR I = 1 TO 3500: NEXT : GOTO 100
90 AD = 43380:B = 43582:C = 202:A = AD
100 CHR =  PEEK (A):A = A + 1
110 B$ =  CHR$ (CHR):A$ = A$ + B$
120  IF CHR > 127 THEN  GOSUB 140: IF A = B THEN 320
130  GOTO 100
140  HOME
150  VTAB 6:TB = (20 - ( LEN (A$) / 2)): HTAB TB: INVERSE : POKE
     33,40 - TB: POKE 32,TB - 1: PRINT A$: TEXT
155  VTAB 23: HTAB 10: NORMAL : PRINT "PRESS (ESC) TO QUIT":
     INVERSE
160  VTAB 10: NORMAL : PRINT "IS THIS THE MESSAGE YOU WANT TO
     CHANGE ?": PRINT "(Y/N) ";: GET AN$
170  IF  ASC (AN$) = 27 THEN 420
180  IF AN$ <  > "Y" THEN A$ = "": RETURN
190  VTAB 16: HTAB 1: PRINT "ENTER->";
200  INVERSE :TB = (20 - ( LEN (A$) / 2)): HTAB TB: FOR I = 1 TO
     LEN (A$): PRINT " ";: NEXT
```

```
210  VTAB 16: HTAB TB
220  FOR I = 1 TO  LEN (A$): GET L$: IF  ASC (L$) = 13 THEN 225
222  PRINT L$;:M$ = M$ + L$: GOTO 227
225  CALL  - 868: PRINT L$;:M$ = M$ + L$: HTAB TB: FOR RE = 1 TO
     ( LEN (A$) -  LEN (M$)): PRINT " ";: NEXT : HTAB TB
227  NEXT
230  NORMAL : PRINT : PRINT : PRINT "ANY CORRECTIONS ? (Y/N) ";:
     GET AN$
235 CV =  PEEK (37)
240  IF AN$ = "Y" AND CV = 17 THEN M$ = "": HTAB 1: CALL  - 868:
     GOTO 190
245  IF AN$ = "Y" AND CV > 17 THEN  VTAB 16: HTAB TB: CALL  -
     958:M$ = "": GOTO 190
250 AC = A -  LEN (M$) - 1
260  FOR I = 1 TO  LEN (M$)
270 K$ =  MID$ (M$,I,1)
280  POKE (AC + I), ASC (K$)
290  NEXT
300  POKE AC + I - 1, ASC (K$) + 128
310 A$ = "":M$ = "": RETURN
320  HOME : VTAB 6: PRINT "DO YOU WANT TO CREATE A BINARY FILE
     TO  SAVE THESE CHANGES ?"
330  PRINT
340  PRINT "(YOU CAN THEN <BLOAD B("N$")REWRITE> IN    HELLO
     PROGRAMS ALREADY ON DISK OR ->"
350  PRINT
360  PRINT "YOU CAN <INIT> A NEW DISK NOW AND THESE CHANGES WILL
     BE PERMANENT)"
370  PRINT : PRINT "(Y/N) ";: GET AN$
380  IF AN$ <  > "Y" THEN 420
390  PRINT
400  PRINT  CHR$ (4);"BSAVE B("N$")REWRITE,A"AD",L"C"
410  PRINT : PRINT "DONE": FOR I = 1 TO 2000: NEXT
420  PRINT : PRINT : PRINT "TRY AGAIN ? (Y/N) ";: GET AN$: IF
     AN$ = "Y" THEN A$ = "":M$ = "": GOTO 20
430  HOME : END
440  HOME : VTAB 6: HTAB 7: PRINT "PERSONALIZED DOS ERROR CODE"
450  VTAB 8: HTAB 18: PRINT "AND"
460  VTAB 10: HTAB 12: PRINT "COMMAND REWRITER"
470  VTAB 16: HTAB 18: PRINT "BY"
480  VTAB 18: HTAB 9: PRINT "DONALD W MILLER JR MD"
490  FOR I = 1 TO 3500: NEXT
500  RETURN
```

Practical Sort for Beginners

Add a simple selection sort routine to enhance your programs. A sample program shows how simple it is.

by JoAnn Miner

How would you go about alphabetizing a list of words? You might start with this simple routine:

1. Pick out the first (lowest) word and write it in the list.
2. Find the next word and write it in the list.
3. Repeat step 2 until all the words are in order.

This is the essence of a simple selection sort routine. It may not be the most efficient or elegant method, but it is easy to understand and it works.

UNDERSTANDING THE FLOW

How can you put this into a program that does something useful? It can be used as a subroutine in a larger program to handle files for your library or record collection. Or it can be used where an ordered list is needed, whether it is alphabetic or numeric.

The algorithm works through the array just as you might check through a list. It looks for the first element, then the second, and so on until the complete array has been checked and put into order.

The names are stored in array A$(N) with N elements. (You could just as easily use numeric data stored in a floating point array A(N).) Start with a FOR-NEXT loop with index I to step through the array one element at a time until the second to last element. This loop sets the elements in their proper order in the new array.

A second FOR-NEXT loop with index J will start at J=I+1 and step through to the end of the array. this loop searches through the remaining elements of the array to find the next in order. If A$(I) is less than A$(J), the elements are in proper order so the program proceeds to the next J. If this test fails it then exchanges these two elements and continues.

Follow these steps to make the exchange:

1. Set a temporary variable, T$=A$(I)
2. Let A$(I)=A$(J)
3. Let A$(J)=T$

By exchanging the positions of the array elements in this way, only one array is needed. This can be a useful feature when you sort long arrays.

Stepping Through the Array

The J loop steps through the array one element at a time until it reaches the last element. Then I is increased by one to the next element of the array. This process continues — J indexing inside the I loop— until I reaches the second to last element of the array. At this point, all the elements have been compared and put into proper position in the list.

A SAMPLE PROGRAM

Listing 12 is a sample program that alphabetizes an array of five words. The size of the array can easily be increased by increasing the dimension in **line 20**. The upper limits of the FOR-NEXT loops in **lines 35, 110, 120 and 210** would also need to be increased.

HOW IT WORKS

Lines 30-50 are a loop to let you enter the list into memory. The sort routine actually starts in **line 100** with the FOR-NEXT loop to index through I from the first to the second to last elements of the array. In **line 120**, a second FOR-NEXT loop is set to index J inside the first loop. J ends on the last element of the array. In this way, the I loop steps the search through the array one element at a time, and the J loop picks elements from the remaining portion of the array to compare with it.

A$(I) is compared to A$(J) in **line 130**. If the Ith element is less than or equal to the Jth element, then these are in order and the next element will be compared. If this test fails, these two elements will be exchanged and the search continues to the next Jth element. In this way, the lower elements are selected and moved to their proper location as the Ith element of A$(I). **Lines 200-240** print out the sorted array and the job is finished.

The sample list of names and printout of the steps though the sort show how this technique works. The program picks the first element of the array and puts it into the first position. On each pass through the remaining elements, the next element is picked. This continues until all elements have been checked.

ON A CLEAR DAY YOU CAN SORT FOREVER

The selection sort is straightforward, but is not very efficient. It does not take advantage of any alphabetic order that might be in the original array. For an array of n elements, the I loop will be repeated n-1 times. For each step if I, the J loop will be repeated n-(I+1) times. For each repetition of the J loop, a comparison will be made. This means that for any sort the number of comparisons is:

$$n+(n-1)+(n-2)+\ldots.+1=(n*(n+1))/2$$

Therefore, for a list of 1000 names, 500,000 comparisons must be made.

This sort method is slow because it doesn't use any information from the list itself. If only the last element of the array is out of place, it does just as much work as if the order is totally random. There are other methods that will work faster in some situations. If you find that the selection sort is just too slow, investigate more advanced methods such as the bubble sort, Quicksort or the Shell-Metzner sort.

LISTING 12: SAMPLE.SORT

```
10   REM  SAMPLE SELECTION SORT PROGRAM
15   REM   LONGER ARRAYS CAN EASILY BE HANDLED
20   DIM A$(5)
30   REM INPUT THE ARRAY
35   FOR I = 1 TO 5
40   INPUT "ENTER A NAME? ";A$(I)
50   NEXT I
100   REM  THE SORT ROUTINE
110   FOR I = 1 TO 4
120   FOR J = I + 1 TO 5
130   IF A$(I) <  = A$(J) GOTO 160
135   REM   EXCHANGE ELEMENTS
140 T$ = A$(I)
145 A$(I) = A$(J)
150 A$(J) = T$
160   NEXT J
170   NEXT I
```

```
200  REM  PRINT OUT THE ARRAY IN ORDER
210  FOR I = 1 TO 5
220  PRINT I,A$(I)
230  NEXT I
240  PRINT : PRINT "THE SORT IS NOW COMPLETE"
999  END
```

Apple Slot Finder

Set up your programs so that they automatically identify which peripherals are in which slots. The program can then select the correct slot without further prompts.

by Steven Weyhrich

Apple Slot Finder is a modification of CONFIG, a program from in the October 1979 issue of CONTACT, the Apple User's Group newsletter. CONFIG checks the Apple's peripheral slots to determine what devices are plugged into them. Recently, I had been working on a program that I wanted to run on Apples with different printer and slot configurations. I wanted the program to automatically select the slot to which the printer was connected.

Slot Finder will identify the Apple Silentype printer card, the Apple Serial Interface card, the Apple Disk II Controller card and Hayes Micromodem II card. For those with an Apple /// running in Apple II emulation mode, it will also identify the Emulation Communications card and the Emulation Serial Interface card.

EMPTY SLOTS

A PR#*s* (where *s* is slot 1-7) to an empty slot causes the computer to hang until the <RESET> key is pressed. An IN#*s* to an empty slot gets you into the Monitor. The Apple tries to redirect its output hooks (with PR#*s*) or its input hooks (with a IN#*s*) to the program that starts at memory address $Cs00. If no peripheral card is plugged in, there is no ROM or RAM at the memory location and the computer crashes. To avoid this, it is useful to know which slots are empty. One way to do this is to have the program execute some lines like:

```
100 INPUT "WHICH SLOT FOR PRINTER?";SL
110 PRINT CHR$(4);"PR#"SL
```

This requires the user to know to which slot the printer is connected. Alternately, the program can have a file on disk that contains the number of the printer slot, and the computer will appear to automatically select the right slot. The drawback to this is if the file was set up for a differently configured Apple, the program will crash when it tries to execute a PR# to an empty slot.

USING SLOT FINDER

When you run Slot Finder (**Listing 13**), if your Apple has any of the above mentioned peripheral cards plugged into it, the program will identify the card as it scans each slot. If you have a card that the program does not allow for, the program will declare that slot empty, even if there really is a card there. (Later, I'll explain how to use the Byte Finder subroutine (**Listing 14**) for the specific configuration of your Apple.)

The Slot Finder subroutine first examines each page of slot memory to determine the presence or absence of a card. To do this, it jumps across the memory of each 256-byte slot and PEEKs at the value of every 64th byte, summing these four bytes into the array CS (checksum). This is done three times. On the Apple II, memory locations without RAM or ROM return pseudo-random numbers when PEEKed. To view this for yourself, enter the Monitor and list part of the memory of an empty slot. For example, for slot 7, use the following command:

```
CALL -151
*C700.C71F
```

Now do it again. Notice that one four-line group is different from the other.

Do the same for a slot that does contain a card, substituting the slot number with the card for each numeral 7. This time the two groups of lines should match, byte for byte at each address. If this is done on an Apple /// in Apple II emulation mode, the numbers returned for an empty slot are all $FF (255). In the Slot Finder subroutine, if the three checksum values match, the slot is occupied; if they don't match, or if the sum reveals each checked byte to be $FF, the slot is empty.

IDENTIFYING THE CARD

Once it finds a filled slot, the next part of the subroutine identifies the card. Since each card has its own unique assembly language routine, comparing the same two or three relative bytes in each different card to known values makes it possible to identify the card.

Byte Finder (**Listing 14**) aids in locating these unique bytes. The addresses used in the original CONFIG program were the fifth and seventh bytes of each card. Since that will not differentiate between an Apple Serial Interface card and a Silentype printer card, I used the tenth and fifteenth bytes of each slot. The subroutine uses a seven-element array called SLOT, and after RETURNing, each element of SLOT holds a number representing the card found in that slot. If no match was found, a zero is returned. If the variable SLOT (4) = 5, slot 4 of your Apple contains a Hayes Micromodem II, the fifth device defined in the initialization part of the driver program.

CONFIGURING YOUR SYSTEM

To make your own program drive the Slot Finder routine, several variables must be set before doing a GOSUB to it. The variables C000, C100 and C700 contain decimal values for the hex numbers $C000, $C100 and $C700, respectively. N represents the number of cards you will define for your program to identify. R1 and R2 represent the relative bytes being checked in each slot.

Four arrays are used by the subroutine: B1 and B2, each dimensioned to size N, hold the known bytes at relative addresses R1 and R2, respectively for each card; CS, the checksum array; and SLOT, the array that, after returning from the subroutine, holds the information gathered by it.

The string array NAME$ is not used by the subroutine. It is used in this example to list the cards identified by the subroutine.

Byte Finder (**Listing 14**) facilitates the identification of bytes and relative addresses unique to your Apple peripheral cards. It will ask for which relative byte to display, and then do a continuously updated listing of that relative byte for all seven slots. When you run it on an Apple II, the empty slots will have different bytes each time the slots are scanned. If you run it on an Apple /// in emulation mode, the empty slots will show with a 255.

MODIFICATIONS

Slot Finder contains no GOTOs or GOSUBs and so is completely relocatable. If you are sure that you won't be using it on an Apple ///, the last three comparisons in the IF statement in **line 420** may be deleted.

Line 360 is the FOR-NEXT loop that skips over the slot memory. Now, every 64th byte is PEEKed. If you want to be more certain that the routine is identifying empty slots, the STEP value in this line can be decreased. However, this will increase the time the routine takes to check all seven slots, since it does more PEEKing. Conversely, if you need more speed, you can increase the STEP value to a number greater than 64. Just be sure that it is still working in the altered mode before you use it in your prize program.

You can also alter the way that Slot Finder saves what it learns about the slot configuration. In its present form, it identifies slots by which cards (if any) are plugged into them. By making the following changes to **Listing 13**, Slot Finder will instead identify cards by the slots they are in. In **line 80**, change SLOT(7) to CARD(N) and **in line 380**, replace SLOT(SLOT) = I with CARD(I) = SLOT. In addition, replace the following lines:

```
240 FOR I = 1 TO N
250 PRINT "THE NAME$(I)"IS ";
260 IF CARD(I) = 0 THEN PRINT "NOT PRESENT"
270 IF CARD(I) < > 0 THEN PRINT "IN SLOT "CARD(I)
310 FOR I = 1 TO N:CARD(I) = 0: NEXT I
```

LISTING 13: SLOT.FINDER

```
10  REM      ***********************
11  REM      *    SLOT.FINDER      *
12  REM      * BY STEVEN WEYHRICH *
13  REM      * COPYRIGHT (C) 1983 *
14  REM      * BY MICROSPARC, INC *
15  REM      * CONCORD, MA. 01742 *
16  REM      ***********************
30  REM   ADAPTED FROM PROGRAM "CONFIG" IN CONTACT #6, APPLE
     USER'S GROUP NEWSLETTER OF OCTOBER 1979
40  REM   IDENTIFIES SLOTS BY WHICH CARDS ARE PLUGGED INTO THEM
50 C000 = 49152:C100 = 49408:C700 = 50944: REM MEMORY ADDRESSES
60 N = 14: REM NUMBER OF CARDS DEFINED
70 R1 = 10:R2 = 15: REM RELATIVE BYTE IN EACH SLOT
80  DIM B1(N),B2(N),NAME$(N),CS(2),SLOT(7)
90 B1(1) = 138:B2(1) = 120:NAME$(1) = "SILENTYPE PRINTER CARD"
100 B1(2) = 120:B2(2) = 072:NAME$(2) = "SERIAL PRINTER CARD"
110 B1(3) = 036:B2(3) = 060:NAME$(3) = "DISK CONTROLLER CARD"
120 B1(4) = 038:B2(4) = 072:NAME$(4) = "COMMUNICATIONS CARD"
130 B1(5) = 255:B2(5) = 007:NAME$(5) = "HAYES MICROMODEM II"
140 B1(6) = 038:B2(6) = 197:NAME$(6) = "EMULATION SERIAL CARD"
150 B1(7) = 0:B2(7) = 84:NAME$(7) = "BUFFERED GRAPPLER PRINTER
     CARD"
160 B1(8) = 0:B2(8) = 85:NAME$(8) = "GRAPPLER PLUS PRINTER CARD"
170 B1(9) = 1:B2(9) = 72:NAME$(9) = "THUNDERCLOCK CARD"
180 B1(10) = 207:B2(10) = 0:NAME$(10) = "SIDER HARD DISK CARD"
190 B1(11) = 6:B2(11) = 151:NAME$(11) = "APPLE SUPER SERIAL
     CARD"
200 B1(12) = 21:B2(12) = 204:NAME$(12) = "APPLE MOUSE CARD"
210 B1(13) = 56:B2(13) = 7:NAME$(13) = "UNIDISK CONTROLLER CARD"
220  GOSUB 310: REM CHECK THE SLOTS
230  REM REPORT ON RESULTS OF SEARCH
240  FOR I = 1 TO 7
250  PRINT "SLOT "I;
260  IF SLOT(I) = 0 THEN  PRINT " IS EMPTY"
270  IF SLOT(I) > 0 THEN  PRINT " HAS A "NAME$(SLOT(I))
280  PRINT : NEXT I
290  END
300  REM  *** SLOT FINDER SUBROUTINE ***
310  FOR I = 1 TO 7:SLOT(I) = 0: NEXT I
320  FOR S = C100 TO C700 STEP 256
330 SLOT = (S - C000) / 256: REM IDENTIFY THE SLOT #
340  REM MAKE 3 PASSES OVER SLOT MEMORY
350  FOR K = 0 TO 2:CS(K) = 0
360  FOR I = 0 TO 255 STEP 64
```

```
370 CS(K) = CS(K) +  PEEK (S + I)
380  NEXT I: NEXT K
390  REM NOW CHECK IF SUM FROM EACH PASS
400  REM IS THE SAME; IF NOT, OR IF ALL
410  REM BYTES ARE $FF, THEN SLOT IS EMPTY
420  IF CS(0) <  > CS(1) OR CS(0) <  > CS(2) OR CS(1) <  > CS(2)
     OR CS(0) = 1020 OR CS(1) = 1020 OR CS(2) = 1020 THEN 470:
     REM EMPTY SLOT
430  REM IDENTIFY THE CARD
440  FOR I = 1 TO N
450  IF  PEEK (S + R1) = B1(I) AND  PEEK (S + R2) = B2(I) THEN
     SLOT(SLOT) = I:I = N: REM A MATCH; TERMINATE LOOP
460  NEXT I
470  NEXT S: REM CHECK THE NEXT SLOT
480  RETURN
```

LISTING 14: BYTE.FINDER

```
10  REM  27-APR-82
11  REM  BYTE.FINDER
12  REM  BY STEVEN WEYHRICH
13  REM  COPYRIGHT(C) 1983
14  REM  BY MICROSPARC, INC
15  REM  CONCORD, MA 01742
40 C000 = 49152:C100 = 49408:C700 = 50944
50 KBD =  - 16384:STR =  - 16368
60  TEXT : HOME
70  FOR K = 0 TO 1:K = 0
80  VTAB 3: PRINT "TYPE ";: INVERSE : PRINT "Q";: NORMAL : PRINT
    " TO QUIT": PRINT
90  INPUT "BYTE # (0 - 255) ";BYTE$
100  IF BYTE$ = "Q" THEN  VTAB 22: END
110 BYTE =  VAL (BYTE$)
120  IF BYTE < 0 OR BYTE > 255 THEN 90
130  VTAB 7
140  PRINT "SLOT  ADDRESS  BYTE #  VALUE"
150  PRINT "----  -------  ------  -----"
160  FOR J = 0 TO 1:J = 0
170  VTAB 10
180  FOR I = C100 TO C700 STEP 256
190 SLOT = (I - C100) / 256 + 1
200 BTE$ =  RIGHT$ ("   " +  STR$ (BYTE),3)
210 VLUE$ =  RIGHT$ ("   " +  STR$ ( PEEK (I + BYTE)),3)
220  PRINT "  "SLOT"    "I"    "BTE$ TAB( 25)VLUE$
230  NEXT I
240  PRINT : PRINT : PRINT "HIT ANY KEY TO CHANGE BYTE #"
250 X =  PEEK (KBD): POKE STR,0: IF X > 127 THEN J = 1
260  NEXT J
270  NEXT K
```

Exec Mini-Assembler

Easily access (under DOS 3.3) the mini-assembler built into ROM and still restore your programs' variable pointers to their original locations.

by Bill Parker

The Apple II has a mini-assembler tucked away in the upper reaches of ROM that is a very limited, scaled-down version of the fancier types of assemblers that are available commercially. It allows you to program in assembly language by translating the mnemonic opcodes (operation codes and operands) that you type into memory into hexadecimal (machine language) object code. The mini-assembler stores the code, line by line, wherever in memory you specify. A good description of it can be found on pp. 49-51 and 66 of the Apple // Reference Manual.

ACCESSING THE MINI-ASSEMBLER

I occasionally venture into the mini-assembler from Applesoft (ROM version) to write a quick-and-dirty assembly language routine to speed up or customize part of a main program. However, the problem with the mini-assembler is that it is actually part of Integer BASIC and thus cannot be used when ROM Applesoft is active. So whenever I wanted to temporarily leave an Applesoft program to use the mini-assembler, I first had to save my program, enter Integer BASIC, enter the Monitor, and finally enter the mini-assembler with a $F666G.

When I was finished with the assembler, I had to return to Integer BASIC, reinitialize Applesoft and reload my program. Worse yet, all of the variable pointers from my Applesoft program were reset, losing previously stored data.

THE SOLUTION

With the assembly language program, SAVE.POINTERS (**Listing 15**) and the Applesoft program CAPTURE.ASSM (**Listing 16**), you can create two EXEC files that will get you into and out of the mini-assembler from Applesoft with a minimum of hassle. Best of all, your original program is restored with its variable pointers returned to their original locations. This provides complete retention of data.

USING THE PROGRAMS

After booting your System Master disk (which loads Integer BASIC into the upper 16K of your Apple), load your Applesoft program. When you're ready to go to the mini-assembler, just type EXEC MINI.ASSM and you're there.

When you are finished with the assembler, press the <RESET> key (or whatever works for you) to return to Integer BASIC. Then type EXEC FP to go back to Applesoft with your program (and variables) fully restored and waiting for you to pick up where you left off.

ENTERING THE PROGRAMS

Key in the program in **Listing 15**, and save it to disk with the command:

BSAVE SAVE.POINTERS,A$300, L$5A

Then type in **Listing 16** and save it with the command:

SAVE CAPTURE.ASSM

This program creates two text files, MINI.ASSM and FP that, when executed, cause the actual use of the assembly language pointer Save and Restore routine that you just typed in.

When you have saved both programs, RUN CAPTURE.ASSM and then delete it (you won't need it anymore). What remains are the two text files, MINI.ASSM and FP, and the assembly language program.

HOW IT WORKS

The first half of SAVE.PTRS (**Listing 15**) saves Applesoft pointers and program bytes while the second half restores them (**see Table 3**).

TABLE 3: Applesoft Pointers

Pointer Name	Function	Hex Location	Decimal Location
1. TXTTAB	Beginning of A/S program	$67,68	103,104
2. VARTAB	Beginning of simple vars, str. ptrs	$69,6A	105,106
3. ARYTAB	Beginning of array vars, str arr ptrs	$6B,6C	107,108
4. STREND	End of memory used by prog vars	$6D,6E	109,110
5. FRETOP	End of free memory (before str)	$6F,70	111,112
6. PRGEND	End of A/S program	$AF,B0	175,176
7. --	Beginning of next statement	$801,802	2049,2050

The program was created to run in Page 3 of memory. It is, however, free of all JMPs and JSRs (which use absolute addressing) and will therefore run in any memory area where there are 90 bytes available. (If you load it into a different memory area, change the CALL statement in CAPTURE.ASSM accordingly.)

When you enter the Monitor or the mini-assembler from Applesoft, your Applesoft program is still in memory. When you return to Applesoft with the FP command (not the EXEC FP text file you just created), Applesoft writes over the first two program bytes with zeros and resets the program pointers to their beginning values. Barring these minor changes, however, your program is still in memory.

The EXEC MINI.ASSM runs the first half of your assembly language routine which saves your Applesoft program and variable pointers while you are still in Applesoft. It then transfers you to Integer BASIC and initializes the mini-assembler. When you return to Integer BASIC from the mini-assembler, you can EXEC FP, which puts you back into Applesoft and calls the second half of the assembly language routine that restores the program and variable pointers. Your program is then exactly the way you left it.

LISTING 15: SAVE .POINTERS

```
 THE ASSEMBLER 1.0

SOURCE FILE -

            ;
            ;************************************
            ;   SAVE.POINTERS
            ; ROUTINE TO SAVE AND RESTORE
            ; APPLESOFT PROGRAM POINTERS
            ;    BY BILL PARKER
            ; (C) 1983 BY MICROSPARC INC.
            ;************************************
            ;
            ; LABELS
```

```
                    PTRTABL  EQU $67
                    PRGEND   EQU $AF
                    PRGBYT   EQU $801
                    ;
                    INIT     ORG $300
                    ;
0300  A2 00         SAVE     LDX #0
0302  B5 67         SAVEPTR1 LDA PTRTABL,X
0304  9D 4C 03               STA PTRSAVE,X
0307  E8                     INX
0308  E0 0A                  CPX #10
030A  D0 F6                  BNE SAVEPTR1
030C  A5 AF         SAVEPTR2 LDA PRGEND
030E  9D 4C 03               STA PTRSAVE,X
0311  E8                     INX
0312  A5 B0                  LDA PRGEND+1
0314  9D 4C 03               STA PTRSAVE,X
0317  E8                     INX
0318  AD 01 08               LDA PRGBYT
031B  9D 4C 03               STA PTRSAVE,X
031E  E8                     INX
031F  AD 02 08               LDA PRGBYT+1
0322  9D 4C 03               STA PTRSAVE,X
0325  60                     RTS
                    ;--------------------------
0326  A2 00         RESTORE  LDX #0
0328  BD 4C 03      GETPTRS1 LDA PTRSAVE,X
032B  95 67                  STA PTRTABL,X
032D  E8                     INX
032E  E0 0A                  CPX #10
0330  D0 F6                  BNE GETPTRS1
0332  BD 4C 03      GETPTRS2 LDA PTRSAVE,X
0335  85 AF                  STA PRGEND
0337  E8                     INX
0338  BD 4C 03               LDA PTRSAVE,X
033B  85 B0                  STA PRGEND+1
033D  E8                     INX
033E  BD 4C 03               LDA PTRSAVE,X
0341  8D 01 08               STA PRGBYT
0344  E8                     INX
0345  BD 4C 03               LDA PTRSAVE,X
0348  8D 02 08               STA PRGBYT+1
034B  60                     RTS
                                             ;RESERVE SPACE FOR POINTERS
                    PTRSAVE  DFS 14          ; ON PAGE 3

000  ERRORS

0300  HEX START OF OBJECT
0359  HEX END OF OBJECT
005A  HEX LENGTH OF OBJECT
9595  HEX END OF SYMBOLS
```

LISTING 16: CAPTURE.ASSM

```
100  REM  CAPTURE.ASSM
110  LET D$ =  CHR$ (4): REM  CTRL-D OR DISK COMMAND
120  PRINT D$;"OPEN MINI.ASSM"
130  PRINT D$;"WRITE MINI.ASSM"
140  PRINT "BRUN SAVE.POINTERS"
150  PRINT "INT"
160  PRINT "CALL -2458"
170  PRINT D$;"CLOSE MINI.ASSM"
180  REM
190  PRINT D$;"OPEN FP"
200  PRINT D$;"WRITE FP"
210  PRINT "FP"
220  PRINT "CALL 806"
230  PRINT D$;"CLOSE FP"
```

Visi-Sort Plus

The speed of the bubble, shell and Quicksort methods are illustrated in these graphics demonstrations.

by Andre Samson

I've adapted the Visi-Sort program by Bill Fortenberry (*Nibble* Vol. 3/No. 5) which gives a graphics demonstration of bubble sorting to two other sort methods. I find them equally fun to watch. As a high school computer science teacher, I agree that the bubble sort is easy to teach. Now, with this visual approach, I would not hesitate presenting the shell and Quicksort methods. My compliments to C. Bongers for his flowchart of the Quicksort algorithm (*Nibble* Vol. 3/No. 4) which is easily coded in Applesoft.

SHELL AND QUICKSORT

More interesting to watch is the worst case run of these three sorts (i.e., ordering on the alternate diagonal/). By simply deleting the shuffling (**line 130**) and reversing the inequality symbol (bubble: **line 10030**; shell: **line 10040**; Quicksort: **lines 10020 and 10030**), you can quickly see the advantages of the latter two sorts. With Quicksort, this diagonal is immediately obtained while the bubble sort is painfully slow to watch.

While the value of the shell and Quicksort methods is most apparent with voluminous data, I recommend maintaining N=100. Most microcomputers will handle this Hi-Res number, so the dimensioning of arrays F,L in Quicksort is unnecessary. Timing these sorts is a bit misleading due to the overhead HPLOTting involved.

LISTING 17: SHELLSORT

```
10  REM  SHELLSORT
100  HGR : HCOLOR= 3
110  DIM A(100): HOME : VTAB 22: PRINT "CREATING ARRAY": FOR I =
     1 TO 100:A(I) = I: NEXT
120  HGR : FOR I = 1 TO 100: HPLOT I,A(I): NEXT
130  HOME : VTAB 22: PRINT "SHUFFLING": FOR I = 1 TO 100:B =
     INT ( RND (1) * 100) + 1:T = A(B):A(B) = A(I):A(I) = T:
     NEXT
140  HGR : FOR I = 1 TO 100: HPLOT I,A(I): NEXT
150 N = 100
160  HOME : VTAB 22: PRINT "METZNER SORT": PRINT ""
170  GOSUB 10000
180  PRINT "": HOME : VTAB 22: PRINT "DONE": END
9000  REM  METZNER (SHELL) SORT"
9001  REM  A ARRAY TO SORT
9002  REM  N ELEMENTS IN ARRAY
10000 M = N
10010 M =  INT (M / 2)
10015  VTAB 23: PRINT "M=";M;" "; CHR$ (7)
10020  IF M = 0 THEN  RETURN
10030  FOR X = 1 TO N - M:H = X
10040 V = H + M: IF A(H) < A(V) THEN 10100
10050 T = A(H):A(H) = A(V):A(V) = T
10060  HCOLOR= 0: HPLOT H,0 TO H,100: HCOLOR= 3: HPLOT H,A(H)
```

```
10070  HCOLOR= 0: HPLOT V,0 TO V,100: HCOLOR= 3: HPLOT V,A(V)
10080 H = H - M
10090  IF H > = 1 THEN 10040
10100  NEXT X
10110  GOTO 10010
```

LISTING 18: QUICKSORT

```
10  REM  QUICKSORT
100  HGR : HCOLOR= 3
110  DIM A(100): HOME : VTAB 22: PRINT "CREATING ARRAY": FOR I =
     1 TO 100:A(I) = I: NEXT
120  HGR : FOR I = 1 TO 100: HPLOT I,A(I): NEXT
130  HOME : VTAB 22: PRINT "SHUFFLING": FOR I = 1 TO 100:B =
     INT ( RND (1) * 100) + 1:T = A(B):A(B) = A(I):A(I) = T:
     NEXT
140  HGR : FOR I = 1 TO 100: HPLOT I,A(I): NEXT
150 N = 100
160  HOME : VTAB 22: PRINT "QUICKSORT": PRINT ""
170  GOSUB 10000
180  PRINT "": HOME : VTAB 22: PRINT "DONE": END
9000  REM  QUICKSORT
9001  REM  A ARRAY TO SORT
9002  REM  N ELEMENTS IN ARRAY
10000 S = 0:F = 1:L = N
10010 M = A( INT ((L + F) / 2)):I = F:J = L
10020  IF A(I) < M THEN I = I + 1: GOTO 10020
10030  IF A(J) > M THEN J = J - 1: GOTO 10030
10040  IF I > J THEN 10110
10050  IF I = J THEN 10090
10060 T = A(I):A(I) = A(J):A(J) = T
10070  HCOLOR= 0: HPLOT J,0 TO J,100: HCOLOR= 3: HPLOT J,A(J)
10080  HCOLOR= 0: HPLOT I,0 TO I,100: HCOLOR= 3: HPLOT I,A(I)
10090 I = I + 1:J = J - 1
10100  IF I < = J THEN 10020
10110  IF I > = L THEN 10130
10120 F(S) = I:L(S) = L:S = S + 1
10130 L = J
10140  IF F < L THEN 10010
10150  IF S = 0 THEN  RETURN
10160 S = S - 1:F = F(S):L = L(S): GOTO 10010
```

LISTING 19: BUBBLESORT

```
10  REM  BUBBLESORT
100  HGR : HCOLOR= 3
110  DIM A(100): HOME : VTAB 22: PRINT "CREATING ARRAY": FOR I =
     1 TO 100:A(I) = I: NEXT
120  HGR : FOR I = 1 TO 100: HPLOT I,A(I): NEXT
```

```
130  HOME : VTAB 22: PRINT "SHUFFLING": FOR I = 1 TO 100:B =
      INT ( RND (1) * 100) + 1:T = A(B):A(B) = A(I):A(I) = T:
      NEXT
140  HGR : FOR I = 1 TO 100: HPLOT I,A(I): NEXT
150 N = 100
160  HOME : VTAB 22: PRINT "BUBBLE SORTING": PRINT ""
170  GOSUB 10000
180  PRINT "": HOME : VTAB 22: PRINT "DONE": END
9000  REM  BUBBLE SORT
9001  REM  A ARRAY TO SORT
9002  REM  N ELEMENTS IN ARRAY
10000 N = N - 0
10010 K = N - 1
10015  FOR I = 1 TO K
10020 L = N - I
10025  FOR J = 1 TO L
10030  IF A(J) < A(J + 1) THEN 10105
10035 T = A(J):A(J) = A(J + 1):A(J + 1) = T
10101  HCOLOR= 0: HPLOT J,0 TO J,100: HCOLOR= 3: HPLOT J,A(J)
10103  HCOLOR= 0: HPLOT J + 1,0 TO J + 1,100: HCOLOR= 3: HPLOT J
      + 1,A(J + 1)
10105  NEXT : NEXT
10120  RETURN
```

Binary Dump

Save paper and ease eyestrain by using this short Applesoft program to print memory dumps in 16-column format.

by Tim Damon

Binary Dump is a small utility that lets you print out a long binary program in 16 columns so you don't have to waste a lot of paper. The printout's chart format makes it easy to find the memory address of any byte by looking at the column headings. The first three numbers of the byte's address are on the left side of each line on the chart and the fourth number is at the top of the column.

Included is a small routine that starts the numbering at the nearest (hexadecimal) number ending in zero. Another routine starts the dump in the appropriate column.

Binary Dump is particularly useful when you are debugging a long program. Once the address of an error is found, you only need to CALL -151, enter the address followed by a colon, and the correct byte.

USING BINARY DUMP

To use Binary Dump, just enter the starting location for the dump and the length of the dump in hexadecimal. To find the starting address of the last binary program loaded off disk, enter the Monitor by typing CALL -151. Then type AA72 and press the <RETURN> key twice. This gives you something like:

```
AA72- 00

03 FF FF FF EF DC 54
```

In this example, the starting address is $300; the 00 follows AA72 and the 03 is the first number on the next line.

To find the program length, enter AA60 and two <RETURN>s. This gives you:

```
AA60- 9C

00 DC 56 5D E7 CE B1
```

or something similar. The program length is $9C, the 9C following AA60 and 00 from the start of the next line.

The Binary Dump program occupies memory locations $800-$FA4 so if you have saved a program that will load in that range, you must put it somewhere else. To do this, enter BLOAD *filename*,A$*nnnn* where *nnnn* is the starting address (somewhere other than $800-$FA4).

ENTERING THE PROGRAM

Key in the program in **Listing 20**, and save it to disk with the command:

SAVE BINARY.DUMP

HOW BINARY DUMP WORKS

Binary Dump uses a PEEK command to get the byte into a variable. (For a list of variables, see **Table 4**.) It then sends the decimal value of the byte through a decimal-hexadecimal conversion and prints it in the chart format. You simply input the beginning memory address and how many bytes to print out.

TABLE 4: VARIABLES

Variable	Function
A$	A work variable for the text centering subroutine and an input variable
BI	A flag to check that the hex inputs are all right
CH	The decimal value of a hex number after going through the hex to decimal subroutine
CH$	The hex value of a decimal number after going through the decimal to hex subroutine
DG	A pointer variable for the decimal to hex conversion subroutine
HX$	A string of valid hex numbers
I, J, M and ZZ	Loop counters
LD	Length of the dump in decimal
NM$ and WK	Work variables for both conversion subroutines
SD	Starting address (in decimal)
W	Tells how many columns to skip over before printing, to start the dump in the appropriate column
X	The decimal value for the starting location to be printed

Binary Dump actually begins at **line 110**; **line 70** jumps to the beginning. **Lines 90 and 100** are one-line subroutines. **Lines 80-87** set NM$ to the decimal value of the location of each line to be printed, jump to the subroutine at **line 700** to convert to hex, then check to be sure that the hex number is four digits long.

Line 90 sets NM$ to the decimal value of the byte to be printed, jumps to the subroutine at line 700 to convert to hex, then prints the byte. **Line 100** is a text-centering subroutine. **Lines 110-230** print the title page and the instructions. Lines 240-290 get the starting location (in hexadecimal) and check for a bad input. **Lines 300-360** get the desired length of the dump (in hexadecimal) and check for a bad input.

Lines 370-390 print the chart line across the printer paper. **Lines 400-460** print the starting address and begin printing in the appropriate column. **Lines 470-550** print the rest of the dump. **Lines 560-600** check if the user wants to quit or get another dump. **Lines 610-690** convert hexadecimal values to decimal and **lines 700-750** convert decimal values to hexadecimal.

LISTING 20: BINARY.DUMP

```
1  REM **********************
2  REM *    BINARY.DUMP     *
3  REM * BY TIM & TOM DAMON *
4  REM * COPYRIGHT (C) 1983 *
5  REM * BY MICROSPARC, INC *
6  REM * CONCORD, MA. 01742 *
7  REM **********************
20  REM  NEXT 3 LINES CHECK FOR PRODOS OR DOS 3.3, THEN CHECK
    FOR LAST BLOAD ADDRESS. IF ABOVE $8000, HIMEM IS SET TO
    $8000 TO PROTECT YOUR CODE FROM STRINGS.
30 PD = ( PEEK (48896) = 76)
40 SA = ( PEEK (43634) + 256 *  PEEK (43635)) *  NOT PD + ( PEEK
    (48855) + 256 *  PEEK (48856)) * PD
50  IF SA > 32768 THEN  HIMEM: 32768
60  HOME
70  GOTO 110
80  PRINT :NM$ =  STR$ (X): GOSUB 700
85  IF  LEN (CH$) < 4 THEN CH$ = "0" + CH$: GOTO 90
87  RETURN
90 NM$ =  STR$ ( PEEK (I)): GOSUB 700: PRINT CH$;" ";: RETURN
100  HTAB 20 -  LEN (A$) / 2 + 1: PRINT A$: RETURN
110  INVERSE
120  HTAB 12: PRINT "******************": HTAB 12: PRINT "*
     *": HTAB 12: PRINT "*                *": HTAB 12: PRINT
     "******************": VTAB 2: REM  16 SPACES IN THE LINES
130  NORMAL
140 A$ = "  BINARY  DUMP  ": GOSUB 100
150 A$ = "BY TIMOTHY DAMON": GOSUB 100: VTAB 12: PRINT "**
     COPYRIGHT 1983 BY MICROSPARC, INC. **"
160  VTAB 23: PRINT "INSTRUCTIONS?": WAIT  - 16384,128
170  GET A$: IF A$ = "N" THEN 240
180  HOME : PRINT "THIS PROGRAM WILL DUMP OUT ANY BINARY
     PROGRAM IN HEXADECIMAL NUMBERS IN 16    COLUMNS."
190  VTAB 8
200  INVERSE :A$ = "HIT ANY KEY TO CONTINUE": GOSUB 100
210  WAIT  - 16384,128
220  NORMAL
230  POKE  - 16368,0
240  HOME : PRINT "ENTER THE STARTING ADDRESS OF THE
     PROGRAM IN HEXADECIMAL.": PRINT "IF YOU DON'T KNOW WHAT IT
     IS THEN TYPE  IN 'END' AND FIND OUT!"
250 HX$ = "0123456789ABCDEF"
260  INPUT "ENTER ADDRESS: ";NM$: IF NM$ = "" THEN 260
270  IF NM$ = "END" THEN  TEXT : HOME : END
280  GOSUB 610: IF BI = 1 THEN  PRINT  CHR$ (7); CHR$
     (7);"INVALID INPUT":BI = 0: GOTO 260
290 SD = CH
300  PRINT : PRINT
310  PRINT "NOW ENTER THE LENGTH IN HEXADECIMAL.": PRINT "IF YOU
     DON'T KNOW THIS THEN TYPE IN      'END' AND FIND OUT!"
320  INPUT "ENTER LENGTH: ";NM$: IF NM$ = "" THEN 320
330  IF NM$ = "END" THEN  TEXT : HOME : END
```

```
340  GOSUB 610: IF BI = 1 THEN  PRINT  CHR$ (7); CHR$
     (7);"INVALID INPUT!":BI = 0: GOTO 320
350 LD = CH - 1
360  PRINT : PRINT
370  HOME
380  PRINT  CHR$ (4);"PR#1":X = SD: GOSUB 80: FOR ZZ = 1 TO  LEN
     (CH$) + 2: PRINT " ";: NEXT : PRINT "0  1  2  3  4  5  6  7
     8  9  A  B  C  D  E  F"
390  FOR ZZ = 1 TO  LEN (CH$) + 2: PRINT " ";: NEXT : PRINT "-
     -  -  -  -  -  -  -  -  -  -  -  -  -  -  -";
400 I = SD:W = SD - 16 *  INT (SD / 16)
410  IF W = 0 THEN 470
420  GOSUB 80:CH$ =  LEFT$ (CH$,3) + "0"
430  PRINT CH$;"- "; SPC( W * 3);
440  FOR I = I TO I + 15 - W
450  GOSUB 90
460  NEXT I
470  FOR X = I TO SD + LD STEP 16
480  GOSUB 80
490  PRINT CH$;"- ";
500  FOR I = X TO X + 15
510  GOSUB 90
520  IF I = SD + LD THEN 550
530  NEXT I
540  NEXT X
550  PRINT : PRINT
560  PRINT  CHR$ (4);"PR#0"
570  PRINT "ANY MORE? (Y/N)";: GET A$
580  IF A$ = "N" THEN  TEXT : HOME : END
590  CLEAR
600  GOTO 240
610 CH = 0:WK = 0
620  FOR M =  LEN (NM$) TO 1 STEP  - 1
630  FOR J = 1 TO 16
640  IF  MID$ (NM$,M,1) =  MID$ (HX$,J,1) THEN CH = CH + (J - 1)
     * 16 ^ WK: GOTO 670
650  NEXT J
660 BI = 1: RETURN
670 WK = WK + 1
680  NEXT M
690  RETURN
700 WK =  VAL (NM$):CH$ = ""
710 DG = WK -  INT (WK / 16) * 16:WK =  INT (WK / 16)
720 CH$ =  MID$ (HX$,DG + 1,1) + CH$
730  IF WK > 0 THEN 710
740  IF  LEN (CH$) < 2 THEN CH$ = "0" + CH$
750  RETURN
```

Hypercounter

Count with the speed of light! this short routine illustrates some of the principles of machine language while it counts to one million, thousands of times faster than Applesoft.

by Ron Macken and Bill Consoli

Are you an Applesoft programmer who is discouraged by the complex logic involved in machine language. Are assemblers hard for you to understand? This fun program illustrates some of the basic logic of machine language and could easily start you on your way to being a successful machine language programmer.

A 70,455% IMPROVEMENT IN SPEED

Try to remember back to the first time you typed in the following program:

```
10 FOR I = 1 TO 1000: VTAB 12: HTAB 18: PRINT I: NEXT I: END
```

When you ran it, weren't you amazed at the speed with which your Apple completed the task? This simple program took about 15.5 seconds to run. If you were amazed by that display of speed, then Hypercounter is just the thing to get your heart pumping. It counts to one million in only 22 seconds. Your Applesoft program would have taken 4 hours, 18 minutes and 20 seconds to accomplish the same feat. That's an 70,455% improvement in speed. Also, as **Listing 21** shows, the logic is pretty easy to understand. It is the basis of all assembly language logic.

The main problem with writing a program like this is that the computer wants to count in hexadecimal instead of decimal. To solve this, we use eight individual digits. The program doesn't actually count; instead it increments the rightmost digit, and if it is a nine, it is set back to zero. The program then increments the digit to the left, also checking that digit.

Another problem is the time it takes to store variables in memory and then print them on the screen. So, instead of counting from 0-9, it counts from $B0 to $B9, which are the ASCII values (in hex) of the digits 0-9. Then the variables are stored directly on the screen. Admittedly, it's a cheap and dirty trick, but it eliminates the time-consuming steps of storing variables in memory, then taking those variables out of memory to print them on the screen. By counting in ASCII and storing the variables right on the screen, simulating a high-speed counter was made a lot easier.

USING THE PROGRAM

To run the program you count just BRUN HYPERCOUNTER, or within your program, include this line:

```
PRINT CHR$(4);"BLOAD HYPERCOUNTER"
```

then simply CALL 768 and watch the numbers fly.

Hypercounter will run for 36 minutes and 40 seconds (the time is takes to count to 99,999,999) or until you press the <RESET> key. Pressing <CTRL>C will not halt the program. CALL 768 will restart the program after you have pressed <RESET>.

ENTERING THE PROGRAM

Please refer to Appendix A for help in entering this program. If you key it in from the Monitor, return to Applesoft, type <CTRL>C<RETURN>, then save it to disk with the command:

BSAVE HYPERCOUNTER,A$300,L$25

HOW HYPERCOUNTER WORKS

First, in **lines 1-9** the eight screen locations ($5B8-$5BF) are assigned a value of $B0, thus putting eight zeros on the screen. The main counting algorithm starts in **line 10**. As each digit is checked, the number of the digit is always stored in Register X, and at the beginning, X is given a value of eight. So the rightmost, or eighth digit is checked to see if it has a value of $B9, or more simply, if it is a nine. Nine times out of ten it isn't, and the program will continue by incrementing the digit (**line 14**), then branching back to **line 10** to start over with the eighth digit.

On that one time when the digit is a nine, the program branches to the KICK routine where the digit is changed from $B9 to $B0 (from a nine to a zero). X is then decremented, thus shifting left one digit, from ones to tens, or tens to hundreds, or hundreds to thousands, etc. Then the program is sent back to CHECK, which performs the same operations on the seventh digit as it did on the eighth. The shift-left procedure continues until a digit is found that is not a nine. At the point, the program goes back to COUNT, which resets X at eight (the rightmost digit) and starts the process over.

LISTING 21: HYPERCOUNTER

```
 0                        ;
 1                        ;
 2                        ; HYPERCOUNTER
 3                        ;
 4                                ORG $300
 5                        SCRN    EQU $5B8        ;FIRST LOCATION ON THE SCREEN
 6                        HOME    EQU $FC58
 7  0300  20 58 FC                JSR HOME          ;CLEAR SCREEN
 8  0303  A9 B0                   LDA #$B0        ;ASC (IN HEX) OF THE DIGIT 0
 9  0305  A2 08                   LDX #8
10  0307  9D B8 05        NUMCL   STA SCRN,X        ;SET XTH DIGIT TO 0
11  030A  CA                      DEX
12  030B  D0 FA                   BNE NUMCL   ;GOTO NUMCL IF ALL 8 DIGITS ARE
13  030D  A2 08           COUNT   LDX #8
14  030F  A9 B9           CHECK   LDA #$B9          ;WILL THIS DIGIT BE GREATER
15  0311  DD B8 05                CMP SCRN,X        ;THAN 9 IF WE INCREMENT IT?
16  0314  F0 06                   BEQ KICK   ;YES? THEN IT NEEDS TO BE KICKED
17  0316  FE B8 05                INC SCRN,X     ;ACTUAL COUNTING IS DONE HERE
18  0319  4C 0D 03                JMP COUNT         ;START OVER
19  031C  A9 B0           KICK    LDA #$B0       ;KICKS DIGIT OVER FROM 9 TO 0
20  031E  9D B8 05                STA SCRN,X     ;PUTS THE DIGIT ON THE SCREEN
21  0321  CA                      DEX
22  0322  D0 EB                   BNE CHECK         ;GO BACK AND DO IT AGAIN
23  0324  60                      RTS               ;WE'VE REACHED 99,999,999 !

000  ERRORS

0300  HEX START OF OBJECT
0324  HEX END OF OBJECT
0025  HEX LENGTH OF OBJECT
95D2  HEX END OF SYMBOLS
```

Custom Catalog

Personalize your DOS 3.3 disks with this Applesoft utility. Easily make changes in the disk header, locked file and file type symbols, file size and even file names.

by Mason Jones

Have you ever wished you had a way to title your disk catalogs? How about simply customizing the catalog's characters, or disguising things on a disk that you want to protect? Custom Catalog will do all that for you — making the process much easier than going into the Monitor to make all the changes you want.

USING CUSTOM CATALOG

Custom Catalog (**Listing 22**) has five main functions. Select the function you want by pressing the number next to it. After each function, or after cancelling a function, you will be asked if you intend to make any other catalog refinements. If you type a Y, you will be returned to the menu. Otherwise, the program will catalog the disk and end. Note that throughout the program, you have to press <RETURN> only when entering more than one character.

Change Headings

This function allows you to change the DISK VOLUME heading at the top of the catalog. This lets you put a title on the disk, possibly indicating whose disk it is or describing what is on the disk. The only restriction is that the title is limited to12 characters or less. Entering longer strings could cause problems when cataloging the disk. If you wish to abort, press <CTRL>Q and then <RETURN>.

This will take your string and POKE the ASCII code of the character with the high bit set (shown on the chart on page 15 of the *Apple Reference Manual*) backwards into locations 45999-46010. This is one of the odd things about DOS — it changes the heading backwards.

Change Lock Symbol

This function lets you choose a character to replace the asterisk for denoting locked files. It requests the ASCII code (high bit set, as above) in case the user wants to enter some nonaccessible characters. It could easily be changed to simply allow input of the character itself, as in the Change Heading function. Again, use <CTRL>Q to quit.

Change Type

Similar to Change Lock Symbol, the Change Type function is used to change the character for denoting the various file types in the catalog. The program will ask you for the ASCII code (high bit set) of the intended character. If you do not want to change the character, simply press <RETURN> without typing anything else.

Change Sizes

Included more for fun than anything else, this will either change all file sizes to be printed as 000, or simply knock off the size-printing routine so the catalog does not show the sizes. This function can be used to make the catalog shorter which is useful for some two-column catalog programs. To abort, press <RETURN>.

Change Names

Similar to Change Sizes, above, this has two options: have no names printed on a catalog, or have the names scrambled. With these two options, you can make it difficult to break into a disk you want to protect.

Making the Changes Permanent

If you happen to do the wrong thing, simply reboot and everything will go back to normal. If you want to make the changes permanent, make the adjustment you want. Then initialize a blank disk. The newly-initialized disk will have all the changes you made right on it, whenever you boot it.

CUSTOMIZING CUSTOM CATALOG

Since Custom Catalog is structured with each function in its own module, adding or changing it should be quite easy. Every function uses the same general form, and exits via the subroutine at **line 820**.

To add your own function, simply add it to the menu portion, and have the GOTO access the proper address. The decimal addresses for each location, in summary, are as follows:

Disk Volume Heading: 45999-46010
Lock Symbol ASCII Code: 44515
Applesoft Type Symbol: 45993
Integer Type Symbol: 45992
Binary Type Symbol: 45994
Text File Type Symbol: 45991

LISTING 22: CUSTOM.CATALOG

```
10  REM  **********************
20  REM  *    CUSTOM.CATALOG    *
30  REM  *    BY MASON JONES    *
40  REM  * COPYRIGHT (C) 1983 *
50  REM  * BY MICROSPARC, INC *
60  REM  * CONCORD, MA. 01742 *
70  REM  **********************
80  REM   ***CATALOG***
90  TEXT : HOME : PRINT "*********** CUSTOM CATALOG ***********"
100  PRINT : PRINT  TAB( 13);"BY MASON JONES": POKE 34,4
110  VTAB 20: PRINT "** COPYRIGHT 1983 BY MICROSPARC, INC. **":
     VTAB 6: INVERSE : HTAB 14: PRINT "CATALOG MENU"
120  NORMAL
130  PRINT : PRINT "1] CHANGE HEADING";
140  PRINT  TAB( 20);"2] CHANGE LOCK SYMBOL"
150  PRINT "3] CHANGE TYPE";
160  PRINT  TAB( 20);"4] CHANGE SIZES"
170  PRINT
180  PRINT "5] CHANGE NAMES";
190  PRINT  TAB( 20);"6] QUIT"
200 CH =  PEEK ( - 16384)
210  IF CH > 175 AND CH < 183 THEN 230
220  GOTO 200
230  POKE  - 16368,0
240  IF CH = 176 THEN  TEXT : HOME : END
250  IF CH = 177 THEN 310
260  IF CH = 178 THEN 430
270  IF CH = 179 THEN 520
280  IF CH = 180 THEN 670
290  IF CH = 181 THEN 750
```

```
300  IF CH = 182 THEN  END
310  HOME : PRINT
320  PRINT "WHEN ASKED, PLEASE INPUT THE HEADING"
330  PRINT : PRINT "YOU WISH TO BE SHOWN WHEN THE DISK"
340  PRINT : PRINT "IS CATALOGED. PLEASE DO NOT INPUT"
350  PRINT : PRINT "MORE THAN 12 CHARACTERS, OR IT MAY"
360  PRINT : PRINT "NOT WORK PROPERLY."
370  PRINT
380  INPUT "HEADING: ";HD$
390  IF HD$ =  CHR$ (17) THEN 820: REM  CTL-Q
400  IF  LEN (HD$) < 12 THEN HD$ = HD$ + " ": GOTO 400
410  FOR X = 1 TO  LEN (HD$):H1$ =  MID$ (HD$,X,1): POKE 46011 -
     X,( ASC (H1$) + 128): NEXT X
420  GOTO 820
430  HOME : PRINT
440  PRINT "WHEN ASKED, INPUT THE SYMBOL YOU WANT"
450  PRINT : PRINT "TO TAKE THE PLACE OF THE ASTERISK": PRINT
460  PRINT "FOR DENOTING A LOCKED FILE IN THE": PRINT
470  PRINT "CATALOG OF THE DISK. YOU MUST INPUT": PRINT : PRINT
     "ASCII CODE OF THE CHARACTER": PRINT : PRINT "WITH THE HIGH
     BIT SET.": PRINT
480  PRINT : INPUT "SYMBOL'S ASCII CODE: ";SY$: IF SY$ =  CHR$
     (17) THEN 820: REM  CTL-Q
490 SY =  VAL (SY$)
500  POKE 44515,SY
510  GOTO 820
520  HOME : PRINT
530  PRINT "WHEN ASKED, INPUT ASCII CODE (HIGH ": PRINT
540  PRINT "BIT SET) OF THE SYMBOL YOU WANT TO STAND"
550  PRINT "FOR THE FILE TYPE MENTIONED. SIMPLY": PRINT
560  PRINT "PRESS <RETURN> TO PASS.": PRINT
570  PRINT : POKE 34,14
580  HOME : INPUT "APPLESOFT FILE: ";AF$:AF =  VAL (AF$)
590  HOME : INPUT "INTEGER FILE: ";IN$:IN =  VAL (IN$)
600  HOME : INPUT "BINARY FILE: ";BF$:BF =  VAL (BF$)
610  HOME : INPUT "TEXT FILE: ";TF$:TF =  VAL (TF$)
620  IF AF$ <  > "" THEN  POKE 45993,AF
630  IF IN$ <  > "" THEN  POKE 45992,IN
640  IF BF$ <  > "" THEN  POKE 45994,BF
650  IF TF$ <  > "" THEN  POKE 45991,TF
660  GOTO 820
670  HOME : PRINT
680  PRINT "YOU CAN EITHER:": PRINT
690  PRINT "    1) MAKE SIZES 000"
700  PRINT "    2) HAVE NO SIZES PRINTED"
710  PRINT : PRINT "    CHOICE: ";: GET CH$: PRINT CH$:CH =  VAL
     (CH$)
720  IF CH = 1 THEN  POKE 44615,169: POKE 44616,0
730  IF CH = 2 THEN  FOR X = 44643 TO 44645: POKE X,234: NEXT X
740  GOTO 820
750  HOME : PRINT
760  PRINT "YOU CAN EITHER:": PRINT
770  PRINT "    1) HAVE NO NAMES PRINTED"
780  PRINT "    2) HAVE NAMES SCRAMBLED"
```

```
790  PRINT : PRINT "   CHOICE: ";: GET CH$: PRINT CH$:CH =  VAL
     (CH$)
800  IF CH = 1 THEN  FOR X = 44571 TO 44573: POKE X,234: NEXT X
810  IF CH = 2 THEN  POKE 44542,32: POKE 44543,72: POKE
     44544,249
820  TEXT : HOME : VTAB 12: PRINT "ANY FURTHER CATALOG
     REFINEMENTS? ";
830  GET B$: PRINT B$: IF B$ = "Y" THEN 80
840  HOME : PRINT  CHR$ (4);"CATALOG": END
```

80-Column Magic

Take advantage of the advanced features available on an 80-column Apple IIe, IIc or IIGS. These two machine language routines give you double-width DOS 3.3 catalogs and double-width Monitor listings.

by G. Mark Fabbi

Two simple patches to DOS will improve upon two common functions of the Apple IIe, IIC and IIGS: catalog and disassembly (the LIST command in the Apple Monitor). Both of these are restricted for the same reason — a lack of information on the screen. Now, with the supplied DOS patch, you can more than double the number of file names on the display.

For those who like to play around in the Monitor, Double List (**Listing 23**) will produce two-column, 40-line Monitor listings. Once you start using it, you will never go back to the old 20-line format again.

DOUBLE CATALOG DOS

In order to modify the catalog function in DOS, I needed to know where it resided in memory and how it worked. You can read more about it in *Beneath Apple DOS* by Don Worth and Pieter Lechner.

The catalog function is handled at address $AD98-$AE2E. After initializing and finding the Volume Table of Contents (VTOC) sector, the catalog is ready to be listed. After each file name is printed, a call is made to the routine at $AE2F. This subroutine skips to the next line, tests if the screen is full, and if so, waits for a keypress to resume the catalog.

To enable DOS to print two file descriptions per line, this subroutine call had to be either changed or eliminated. Eliminating the subroutine call scattered file names and sector sizes all over the screen. The output routine had done just what I had told it to — send a stream of characters to the display without any linefeeds.

Since this attempt used all 80 columns for the output, I knew I had to remove the linefeed only after the first file name on each line.

Further research with *Beneath Apple DOS* provided a possible solution. The first line of a table listing zero-page usage held the key. Value $24 contains the "cursor horizontal" position on the screen. By testing this value, it should be possible to determine if one or two file descriptions have been printed on a line.

After some experimenting, I discovered that this counter is not updated with each character displayed on the screen, so that after printing the first file description, its value is still $00. Therefore, my program needed to determine whether the value is a zero, and if it is, do an immediate return to the catalog function. Otherwise, it calls the routine at $AE2F to issue the linefeed and look after the vertical line count.

Now that I had found a solution, I needed space to insert it into DOS — no longer as easy to find as it once was. With the release of the Apple IIe came a small revision to DOS 3.3. This revision corrected bugs in the append and position functions and used much of the free space that was formerly available.

I reworked the section of code that prints the message DISK VOLUME at the top of each catalog. The heading now reads VOL, and the patch overwrites part of the space where the text DISK VOLUME used to be. This method does not interfere with other DOS modifications or detract from any of DOS 3.3's functions.

The following steps are required to use the routine:

1. Boot a disk with normal DOS.
2. Type CALL -151 to enter the Monitor.

3. Type B3AF:A5 24 F0 03 20 2F AE 60. This inserts the patch into DOS.
4. Type AE22:20 AF B3. This diverts operation to the patch.
5. Type ADAF: 04
   ADB1: B7
   B3B7: AE EC EF D6
   This changes the heading to read "Vol."
6. This optional step increases the number of lines printed in the catalog:
   Type ADA4: 17
   AE3D: 17

Once all the changes have been made and carefully tested, you can make the modified DOS a permanent part of your software library by initializing a new disk with the new DOS in memory. The above patch will not in any way affect the normal operation of DOS 3.3. It will still function as it should on any Apple II Plus or 40-column IIe. To use the double-width catalog, the unenhanced Apple IIe must be in 80-column mode. Other users should see 80-Column Catalog on page 133.

Warning: this patch leaves you with a nonstandard DOS. While I have been careful to avoid conflict with other patches and software, you could run into problems. Before implementing this patch with a nonstandard DOS, check that it does not interfere with your previous modifications.

DOUBLE DISASSEMBLY

A different method had to be used to modify the LIST command, since it resides in the Monitor ROM (which cannot be changed without programming a new EPROM). This meant that a new command had to be used to get an assembly listing in two columns.

I used the Monitor's <CTRL>Y command to activate the new function. The <CTRL>Y command forces the Monitor to jump to memory location $3F8, where a JMP instruction can be placed to direct control to your own program.

The first step in using the <CTRL>Y function is to set the jump at $3F8 ($300-30F of **Listing 23**). This routine must be executed before the Double List feature can be used. (An alternate method to install the necessary jump would be to type in the Monitor command 3F8:4C 10 03.)

USING DOUBLE LIST

To use Double List, BRUN DOUBLE LIST to set up the <CTRL>Y pointer. Once this has been done, the function can be activated from the Monitor by typing (addr) <CTRL>Y. This will give you 40 lines of disassembly starting at the specified address. To continue the list, type <CTRL>Y. The routine will remember where it left off.

ENTERING DOUBLE LIST

Please refer to Appendix A for help in entering this program. If you key it in from the Monitor, save it to disk with the command:

```
BSAVE DOUBLE LIST,A$300,L$48
```

HOW IT WORKS

The Double List routine uses a simple trick to manipulate the text window to get two listings on the screen at the same time. The text window is defined by four bytes in zero page (see **Table 5**). These four bytes determine what part of the display is active. Parts of the screen that are not active cannot be written to or scrolled off the screen. By changing the width and left edge of the screen, the second column can be printed on the right side without destroying the left half of the screen.

TABLE 5: Active Text Window Display

	Location Dec	Location Hex	80-Column Range Dec	80-Column Range Hex
Left Edge	32	$20	0-79	$0-$4F
Width	33	$21	0-80	$0-$4F
Top Edge	34	$22	0-23	$0-$17
Bottom Edge	35	$23	0-24	$0-$18

Note: Left Edge plus Width must always be less than the current screen setting (40 or 80).

The routine uses code that is already present in the Monitor and is easy to understand. The first information it needs is the location to start the listing. This function is located at $FFA7 and is called GETNUM. This procedure analyzes the characters in the input buffer, converts them to hex digits, and stores them in a Zero-page Register. The value in the Y-Register tells where in the buffer to start looking.

The subroutine call to $33D transfers the address from the Zero-page Registers that GETNUM uses to the registers that are used by the list function. The 80-column card is then activated by the call to $FE95.

The Apple is now ready to list the first 20 lines. This is done by calling the subroutine at $FE63 called LIST2, an alternate entry point to the LIST command.

Once the first disassembly is done, the text window must be set to allow a second column to be displayed on the screen. After resetting the text window (being careful to change the width before changing the left edge), you are ready to list the next 20 lines. Another call is made to $FE63, and you have a neat two-column, 40-line disassembly.

Before the program returns to the Monitor, the text window must be restored to its original state and the hooks to DOS must be repaired. The last five lines of the routine handle these functions.

Warning: Double List routine works on Apple IIe's with the new Apple 80-column card only. Use of this program on a II Plus or a 40-column IIe will cause an immediate system crash.

MODIFICATIONS

This routine can easily be relocated if you are using the $300 area for another routine. Changes must be made to Listing 1 to let the <CTRL>Y function know where to find the routine, and to modify the JSR $33D instruction to reflect the new location of the subroutine. All other branches in the routine are relative, so they do not need to be changed.

REFERENCES

1. Watson, Allen. *Apple IIe Reference Manual*, Apple Computer, Inc.
2. Ibid. Addendum: "Monitor ROM Listing."
3. O'Shea, Donald C. "The Disassemblist," *Nibble*, Vol.3/No.5.
4. Worth, Don and Pieter Lechner. *Beneath Apple DOS*, Quality Software.
5. Zaks, Rodney. *Programming the 6502*, Sybex.

Listing 23: Double List

```
0300:  A9  4C      LDA  #4C
0302:  8D  F8  03  STA  $3F8   ;Set up <CTRL>Y pointers
0305:  A9  10      LDA  #10
0307:  8D  F9  03  STA  $3F9
030A:  A9  03      LDA  #03
030C:  8D  FA  03  STA  $3FA
030F:  60          RTS
0310:  A0  00      LDY  #0
0312:  20  A7  FF  JSR  $FFA7  ;Get starting address
0315:  20  3D  03  JSR  $33D   ;Move addr to PC
0318:  A9  03      LDA  #3
031A:  20  95  FE  JSR  $FE95  ;PR#3
031D:  A9  14      LDA  #14    ;# of lines to list
031F:  20  63  FE  JSR  $FE63  ;List 20 lines
0322:  A2  27      LDX  #27
0324:  86  21      STX  $21
0326:  86  20      STX  $20    ;Set window
0328:  A2  01      LDX   #1
032A:  86  25      STX  $25    ;Set vertical cursor
032C:  A9  14      LDA  #14    ;# of lines to list
032E:  20  63  FE  JSR  $FE63  ;List 20 lines
0331:  A2  00      LDX  #0
0333:  86  20      STX  $20
0335:  A2  4F      LDX  #4F
0337:  86  21      STX  $21    ;Reset window
0339:  20  EA  03  JSR  $3EA   ;Reconnect DOS
033C:  60          RTS
033D:  8A          TXA
033E:  F0  07      BEQ  $347   ;No addr input
0340:  B5  3E      LDA  $3E,X
0342:  95  3A      STA  $3A,X  ;Move addr to PC
0344:  CA          DEX
0345:  10  F9      BPL  $340
0347:  60          RTS
```

Eleven Free Sectors

Take back the 11 sectors DOS 3.3 allocates, but doesn't use. This short machine language routine lets you write to the unused sectors on track 2.

by Les Stewart

Everybody knows that DOS uses the first three tracks on the disk, right? DOS 3.3 does use tracks 0 and 1, but the Volume Table of Contents (VTOC) only uses the first five sectors (0-4) of track 2. However, it shows all of track 2 as used, so the last 11 sectors are not available to you. Eleven sectors are not much, but they could be the difference between a DISK FULL message and getting that last program on the disk.

Listing 24 shows a short machine language program to free up the 11 sectors. Please refer to Appendix A for help in entering this program. If you key it in from the Monitor, save it to disk with the command:

BSAVE FREE.11.SECTORS,A$300,L$41

Once in place, a CALL 768 from Applesoft or a 300G command from the Monitor will change the VTOC on your disk to show sectors 5-16 on track 2 as free. Check your typing carefully — an error in this program could mess up your disk.

Listing 24: FREE.11.SECTORS

```
                     ;
                     ;************************************
                     ;      Free.11.Sectors
                     ;      by Les Stewart
                     ;  Copyright (C) 1984 by
                     ;     MicroSPARC, Inc.
                     ;
                     ;
                     ;  This program will add eleven
                     ;  usable sectors to a DOS 3.3
                     ;  disk.
                     ;
                     ;************************************
                     ;
                              ORG $300
0300  A9 00                   LDA #$00        ;Initialize
0302  8D ED B7                STA $B7ED       ;Sector no.
0305  8D EB B7                STA $B7EB       ;Vol. no.
0308  8D F3 B7                STA $B7F3       ;Counter for partial sector
030B  8D F0 B7                STA $B7F0       ;Buffer pointer lo
030E  A9 11                   LDA #$11        ;Track no.
0310  8D EC B7                STA $B7EC
0313  A9 40                   LDA #$40        ;Buffer pointer hi
0315  8D F1 B7                STA $B7F1
0318  A9 01                   LDA #$01        ;Set up read
031A  8D F4 B7                STA $B7F4
031D  20 E3 03                JSR $03E3       ;Enter DOS
0320  20 D9 03                JSR $03D9       ;Call RWTS for read
0323  A9 00                   LDA #$00        ;Restore status register
```

```
0325  85 48        STA $48
0327  A9 FF        LDA #$FF        ;Change first two bytes
0329  8D 40 40     STA $4040
032C  A9 E0        LDA #$E0
032E  8D 41 40     STA $4041
0331  A9 02        LDA #$02        ;Set up write
0333  8D F4 B7     STA $B7F4
0336  20 E3 03     JSR $03E3       ;Enter DOS
0339  20 D9 03     JSR $03D9       ;Call RWTS for write
033C  A9 00        LDA #$00        ;Restore status register
033E  85 48        STA $48
0340  60           RTS             ;Done

000  ERRORS

0300  HEX START OF OBJECT
0340  HEX END OF OBJECT
0041  HEX LENGTH OF OBJECT
95FF  HEX END OF SYMBOLS
```

Fancy Hi-Res Picture Loading

Tired of watching the "venetian blinds" open as your Hi-Res picture is loaded? Try these machine language routines for special effects display of either Hi-Res screen.

by Art Arizpe

Add some style and flash to your Hi-Res picture loading with two short machine language programs that let you avoid the venetian blind effect and provide a neat and different way to load your graphics screens.

HOW TO USE THE PROGRAMS

The first routine, Fancy Hi-Res Loader 1, is shown in **Listing 25**. When you load a picture with it, the middle of the picture will appear first, as a horizontal strip. The strip will widen toward the top and bottom of the screen until the entire picture is revealed.

The effect of Fancy Loader 2 (**Listing 26**) is similar to Fancy Hi-Res Loader 1, but the picture first appears as a vertical strip and widens toward the left and right edges of the screen.

Listing 27 shows a short Applesoft program that demonstrates the use of the two Hi-Res loaders. They are very easy to use from BASIC.

Both programs require that you initially load your picture onto either Hi-Res page 1 or page 2. The programs will then transfer the picture to the other (destination) page with the special effect. You tell the programs the page on which your picture is located by POKEing a value into location $2F0 (752 decimal). If the value in $2F0 is equal to zero, the picture is located on page 1 and the program will transfer it to page 2. If the value in $2F0 is anything else, the picture is located on page 2 and the program will transfer it to page 1.

Use these routines from BASIC as follows:

1. Without issuing an HGR or HGR2 command, BLOAD your picture onto the page that is not the destination page.
2. POKE 752,0 if page 2 is the destination page. POKE 752,1 if page 1 is the destination page.
3. CALL 768 to transfer the picture to your destination page.

Using HPOSN to calculate the base address of a Hi-Res line is a simple, but effective way to accomplish quite a bit in a short program. Using HPOSN, you can also write routines that will do a color reversal on a Hi-Res picture or will filter out particular colors from a picture.

ENTERING THE PROGRAMS

Please refer to Appendix A for help in entering these programs. If you key them in from the Monitor, save them with the commands:

BSAVE FANCY.LOADER1,A$300,L$AE

and

BSAVE FANCY.LOADER2,A$300,L$D0

HOW THEY WORK
FANCY HI-RES LOADER 1

The program uses HPOSN, a routine located at $F411 (62481) in Applesoft ROM. HPOSN will position the Hi-Res cursor without plotting. The position of the cursor is passed to HPOSN as: horizontal — Y-Register (MSB), X-Register (LSB); vertical — A-Register.

HPOSN will leave the address of the first byte of the particular Hi-Res line specified by the A-Register in locations $26 and $27 on page zero.

HPOSN not used here to plot anything. I use it to locate the base address of any Hi-Res line. It then becomes a simple matter to use the Y-Register in the post-indexed indirect addressing mode of the 6502 to transfer each of the bytes in a Hi-Res line from one page to the other. This is accomplished in **lines 76–99** of **Listing 25**.

The rest of the program involves determining the correct origin and destination pages, clearing the destination page, and looping through the 192 Hi-Res lines in the right order. Fancy Hi-Res Loader 1 is fairly short (173 bytes) and **Listing 25** is set up to assemble at location $300 (768). The code is relocatable so you can place it anywhere you have room.

FANCY HI-RES LOADER 2

Listing 26 shows the second routine, Fancy Hi-Res Loader 2. At 207 bytes, the program is a bit longer than the other. It will still fit in the space starting at $300, but just barely — the program ends at $3CF and the DOS pointers start at $3D0.

MODIFICATIONS

You can control the speed of the two transfers. Each program calls the routine WAIT at $FCA8 in the Monitor ROM in order to provide a delay to the transfer. This delay is determined in **line 71** in **Listing 25**, and **line 85** in **Listing 26**. Experiment with these values until you find the speed that you like best.

LISTING 25: FANCY.LOADER1

```
SOURCE FILE -

 0                        ;
 1                        ;      FANCY.LOADER1
 2                        ;      WRITTEN BY ART ARIZPE
 3                        ;      (C) 1983 MICROSPARC, INC.
 4                        ;      CONCORD, MA 01742
 5                        ;
 6                        ;
 7                        PTR2     EQU $0         ;DESTINATION POINTER
 8                        PTR1     EQU $26        ;ORIGIN POINTER
 9                        HPAG     EQU $E6   ;PAGE ZERO POINTER, BASE ADDRESS
10                        ;                  ;OF HI-RES LINE
11                        HPOSN    EQU $F411 ;ROM ROUTINE TO LOAD BASE
12                        ;                  ;ADDRESS OF HI-RES LINE
13                        WAIT     EQU $FCA8      ;DELAY ROUTINE
14                        FLAG     EQU $2F0
15                        COUNT    EQU $2F1
16                        FROM     EQU $2F2
17                        Y1       EQU $2F3
18                        Y2       EQU $2F4
19                        ;
20                        ;
21                                 ORG $300
22  0300  AD F0 02                 LDA FLAG  ;DETERMINE IF WE ARE MOVING FROM
23  0303  D0 08                    BNE P2.P1      ;PAGE 1 TO PAGE 2
24  0305  A9 20                    LDA #$20
25  0307  8D F2 02                 STA FROM
26  030A  0A                       ASL
27  030B  D0 06                    BNE CONT1
28  030D  A9 40           P2.P1    LDA #$40       ;OR PAGE 2 TO PAGE 1
```

```
030F  8D F2 02            STA FROM
0312  4A                  LSR
0313  85 E6     CONT1     STA HPAG         ;INITIALIZE VARIABLES
0315  A9 00               LDA #$00
0317  8D F1 02            STA COUNT
031A  A9 5F               LDA #$5F
031C  8D F3 02            STA Y1
031F  8D F4 02            STA Y2
0322  EE F4 02            INC Y2
0325  AD F1 02  NXTLIN    LDA COUNT ;FAST HI-RES ERASE ROUTINE
0328  A2 00               LDX #$00
032A  A0 00               LDY #$00
032C  20 11 F4            JSR HPOSN
032F  A9 00               LDA #$00
0331  A8                  TAY
0332  91 26     NXTBYT    STA (PTR1),Y
0334  C8                  INY
0335  C0 28               CPY #$28
0337  D0 F9               BNE NXTBYT
0339  EE F1 02            INC COUNT
033C  AD F1 02            LDA COUNT
033F  C9 C0               CMP #$C0
0341  D0 E2               BNE NXTLIN
0343  AD F2 02            LDA FROM
0346  85 E6               STA HPAG
0348  A9 60               LDA #$60
034A  8D F1 02            STA COUNT
034D  AD 50 C0            LDA $C050        ;SET GRAPHICS MODE
0350  AD 52 C0            LDA $C052        ;SET FULL SCREEN
0353  AD 57 C0            LDA $C057        ;SET HI-RES MODE
0356  AD F0 02            LDA FLAG
0359  D0 06               BNE HGR1
035B  AD 55 C0            LDA $C055        ;DISPLAY PAGE 2
035E  18                  CLC
035F  90 03               BCC LOOP1
0361  AD 54 C0  HGR1      LDA $C054        ;OR PAGE 1
0364  AE F3 02  LOOP1     LDX Y1           ;MAIN ROUTINE
0367  20 81 03            JSR MOVE
036A  CE F3 02            DEC Y1           ;MOVES A PAIR OF LINES
036D  AE F4 02            LDX Y2
0370  20 81 03            JSR MOVE
0373  EE F4 02            INC Y2
0376  A9 30               LDA #$30    ;THEN A SHORT PAUSE
0378  20 A8 FC            JSR WAIT
037B  CE F1 02            DEC COUNT
037E  D0 E4               BNE LOOP1
0380  60                  RTS              ;RETURN TO CALLER
0381  8A        MOVE      TXA              ;ROUTINE WHICH MOVES A LINE
0382  A2 00               LDX #$00
0384  A0 00               LDY #$00         ;FROM THE ORIGIN PAGE
0386  20 11 F4            JSR HPOSN        ;TO THE DESTINATION PAGE
0389  A5 26               LDA PTR1
038B  85 00               STA PTR2
038D  AD F0 02            LDA FLAG         ;IS PAGE 1 THE ORIGIN?
0390  D0 09               BNE P2
0392  18                  CLC
0393  A5 27               LDA PTR1+1
0395  69 20               ADC #$20
```

```
0397  85 01            STA PTR2+1
0399  D0 07            BNE CONT2
039B  38        P2     SEC                ;OR PAGE 2?
039C  A5 27            LDA PTR1+1
039E  E9 20            SBC #$20
03A0  85 01            STA PTR2+1
03A2  A0 00     CONT2  LDY #$00
03A4  B1 26     LOOP2  LDA (PTR1),Y ;LOOP; TRANSFER BYTES FROM
03A6  91 00            STA (PTR2),Y ;PAGE TO PAGE
03A8  C8               INY
03A9  C0 28            CPY #$28
03AB  D0 F7            BNE LOOP2
03AD  60               RTS

000  ERRORS

0300  HEX START OF OBJECT
03AD  HEX END OF OBJECT
00AE  HEX LENGTH OF OBJECT
956E  HEX END OF SYMBOLS
```

LISTING 26: FANCY.LOADER2

```
SOURCE FILE -

                    ;
                    ;      FANCY.LOADER2
                    ;      WRITTEN BY ART ARIZPE
                    ;      (C) 1983 MICROSPARC, INC.
                    ;      CONCORD, MA 01742
                    ;
                    PTR2    EQU $0         ;DESTINATION POINTER
                    PTR1    EQU $26        ;ORIGIN POINTER
                    HPAG    EQU $E6    ;PAGE ZERO POINTER, BASE ADDRESS
                    ;                  ;OF HI-RES LINE
                    HPOSN   EQU $F411  ;ROM ROUTINE TO LOAD BASE
                    ;                  ;ADDRESS OF HI-RES LINE
                    WAIT    EQU $FCA8      ;DELAY ROUTINE
                    ;
                    FLAG    EQU $2F0
                    FROM    EQU $2F1
                    TEMP    EQU $2F2
                    XCOUNT  EQU $2F3
                    YCOUNT  EQU $2F4
                    X1      EQU $2F5
                    X2      EQU $2F6
                    Y1      EQU $2F7
                    Y2      EQU $2F8
                    ;
                            ORG $300
0300  AD F0 02              LDA FLAG   ;DETERMINE IF WE ARE MOVING FROM
0303  D0 08                 BNE P2.P1      ;PAGE 1 TO PAGE 2
0305  A9 20                 LDA #$20
0307  8D F1 02              STA FROM
030A  0A                    ASL
030B  D0 06                 BNE CONT1
030D  A9 40         P2.P1   LDA #$40       ;OR PAGE 2 TO PAGE 1
```

```
030F  8D F1 02           STA FROM
0312  4A                 LSR
0313  85 E6     CONT1    STA HPAG          ;INITIALIZE VARIABLES
0315  A2 13              LDX #$13
0317  8E F5 02           STX X1
031A  E8                 INX
031B  8E F6 02           STX X2
031E  8E F3 02           STX XCOUNT
0321  A2 5F              LDX #$5F
0323  8E F7 02           STX Y1
0326  E8                 INX
0327  8E F8 02           STX Y2
032A  8E F4 02           STX YCOUNT
032D  A9 00              LDA #$00
032F  8D F2 02           STA TEMP
0332  AD F2 02  NXTLIN   LDA TEMP   ;FAST HI-RES ERASE ROUTINE
0335  A2 00              LDX #$00
0337  A0 00              LDY #$00
0339  20 11 F4           JSR HPOSN
033C  A9 00              LDA #$00
033E  A8                 TAY
033F  91 26     NXTBYT   STA (PTR1),Y
0341  C8                 INY
0342  C0 28              CPY #$28
0344  D0 F9              BNE NXTBYT
0346  EE F2 02           INC TEMP
0349  AD F2 02           LDA TEMP
034C  C9 C0              CMP #$C0
034E  D0 E2              BNE NXTLIN
0350  AD F1 02           LDA FROM
0353  85 E6              STA HPAG
0355  AD 50 C0           LDA $C050         ;SET GRAPHICS MODE
0358  AD 52 C0           LDA $C052         ;SET FULL SCREEN
035B  AD 57 C0           LDA $C057         ;SET HI-RES MODE
035E  AD F0 02           LDA FLAG
0361  D0 06              BNE HGR1
0363  AD 55 C0           LDA $C055         ;DISPLAY PAGE 2
0366  18                 CLC
0367  90 03              BCC LOOP1
0369  AD 54 C0  HGR1     LDA $C054         ;OR PAGE 1
036C  A2 5F     LOOP1    LDX #$5F          ;MAIN ROUTINE
036E  8E F7 02           STX Y1
0371  E8                 INX
0372  8E F8 02           STX Y2
0375  8E F4 02           STX YCOUNT
0378  AE F7 02  LOOP2    LDX Y1            ;MOVES TWO COLUMNS OF BYTES
037B  20 A0 03           JSR MOVE
037E  CE F7 02           DEC Y1            ;BETWEEN THE TWO PAGES
0381  AE F8 02           LDX Y2
0384  20 A0 03           JSR MOVE
0387  EE F8 02           INC Y2
038A  CE F4 02           DEC YCOUNT
038D  D0 E9              BNE LOOP2
038F  A9 30              LDA #$30          ;SHORT PAUSE
0391  20 A8 FC           JSR WAIT
0394  CE F5 02           DEC X1
0397  EE F6 02           INC X2
039A  CE F3 02           DEC XCOUNT
```

```
039D  D0 CD             BNE LOOP1
039F  60                RTS              ;RETURN TO CALLER
03A0  8A        MOVE    TXA        ;MOVE A COLUMN OF BYTES
03A1  A2 00             LDX #$00
03A3  A0 00             LDY #$00
03A5  20 11 F4          JSR HPOSN
03A8  A5 26             LDA PTR1
03AA  85 00             STA PTR2
03AC  AD F0 02          LDA FLAG         ;IS PAGE 1 THE ORIGIN?
03AF  D0 09             BNE P2
03B1  18                CLC
03B2  A5 27             LDA PTR1+1
03B4  69 20             ADC #$20
03B6  85 01             STA PTR2+1
03B8  D0 07             BNE CONT2
03BA  38        P2      SEC              ;OR PAGE 2?
03BB  A5 27             LDA PTR1+1
03BD  E9 20             SBC #$20
03BF  85 01             STA PTR2+1
03C1  AC F5 02  CONT2   LDY X1        ;MOVE BYTES FROM PAGE TO PAGE
03C4  B1 26             LDA (PTR1),Y
03C6  91 00             STA (PTR2),Y
03C8  AC F6 02          LDY X2
03CB  B1 26             LDA (PTR1),Y
03CD  91 00             STA (PTR2),Y
03CF  60                RTS

000  ERRORS

0300  HEX START OF OBJECT
03CF  HEX END OF OBJECT
00D0  HEX LENGTH OF OBJECT
9553  HEX END OF SYMBOLS
```

LISTING 27: FANCY.LOADER.DEMO

```
10  REM **********************
11  REM * FANCY.LOADER.DEMO  *
12  REM *   BY  A. ARIZPE    *
13  REM * COPYRIGHT (C) 1983 *
14  REM * BY MICROSPARC, INC *
15  REM * CONCORD, MA  01742 *
16  REM **********************
70  TEXT : HOME : VTAB 2: PRINT "** COPYRIGHT 1983 BY
     MICROSPARC, INC. **": VTAB 10: HTAB 8: PRINT "1) FANCY HI-
     RES LOADER 1"
80  PRINT : HTAB 8: PRINT "2) FANCY HI-RES LOADER 2"
90  VTAB 24: HTAB 10: INPUT "WHICH LOADER? ";LDR
100  IF LDR = 1 THEN  PRINT  CHR$ (4);"BLOAD FANCY.LOADER1"
110  IF LDR = 2 THEN  PRINT  CHR$ (4);"BLOAD FANCY.LOADER2"
120  TEXT : HOME : VTAB 10: INPUT "PLEASE ENTER THE HI-RES
     PICTURE FILENAME: ";NA$: PRINT
130  INPUT "TO WHICH PAGE DO YOU WISH TO LOAD        THE HI-RES
     PICTURE (1 OR 2)? ";PAGE
140 ADR = 8192:ADR = ADR * (3 - PAGE)
```

```
150  PRINT  CHR$ (4);"BLOAD";NA$;",A";ADR
160  IF PAGE = 1 THEN  POKE 752,1
170  IF PAGE = 2 THEN  POKE 752,0
180  CALL 768: REM TRANSFER PAGE
190  GET A$: GOTO 120
```

Double Hi-Res Graphics for the Apple II Plus

Now you can get higher resolution graphics from your Apple II Plus. All you need is a high resolution monochrome monitor and this short machine language routine.

by Algis J. Matyckas

You don't have to be left out of the double Hi-Res graphics experience just because you own an Apple II Plus. You can plot 560 points horizontally in two colors — black and white. That's twice the resolution provided by the 280 points plotted in normal Hi-Res mode.

First, let me give a brief technical description of Hi-Res graphics. The II Plus has two blocks of memory reserved for Hi-Res graphics. The first block, Hi-Res page 1, is located in memory addresses $2000-$3FFF. The second block, Hi-Res page 2, is located at addresses $4000-$5FFF. Each page is made up of 192 rows that are 40 bytes wide, and each byte is made up of 7 bits, each of which controls a point on the screen. (The eighth bit is not displayed.) So, 40 bytes times 7 bits equals 280 points horizontally.

For a quick Hi-Res demonstration, type:

```
HGR: VTAB 24: CALL -151 <RETURN>
```

This will display Hi-Res page 1, move your cursor to the text window portion of the screen, and put you into the Monitor.

To set the individual bits to plot points on the screen, type:

```
2000:01 <RETURN>
```

You will see a point in the upper-left corner of the screen that results from placing the value $01 in the first byte of Hi-Res screen memory. In this Monitor command, 2000 specifies the memory address in hexadecimal, the 01 is the value of the byte to be written into this address, and the colon (:) is the Monitor command to write byte(s) to the specified location.

Now enter the following sequence:

```
2000:02 <RETURN>
2000:04 <RETURN>
2000:08 <RETURN>
2000:10 <RETURN>
2000:20 <RETURN>
2000:40 <RETURN>
```

If you were to repeat this sequence using the appropriate address for each of the 40 bytes, you would see all 280 points in the top row.

The eighth bit in each byte is called the color bit. Instead of controlling a point, it determines the color of the points plotted by that byte. You can use the color bit to create double Hi-Res graphics. Each of the first seven bits now represents two narrower points on the screen. If the color bit in the byte is clear (set to zero), then the left point is plotted for that bit. If the color bit is set (to one), then the right point is plotted.

Enter the following sequence and watch the display:

2000:01 <RETURN>

```
2000:81 <RETURN>
2000:02 <RETURN>
2000:82 <RETURN>
2000:04 <RETURN>
2000:84 <RETURN>
2000:08 <RETURN>
2000:88 <RETURN>
2000:10 <RETURN>
2000:90 <RETURN>
2000:20 <RETURN>
2000:A0 <RETURN>
2000:40 <RETURN>
2000:C0 <RETURN>
```

If you were to repeat this sequence for 40 bytes, you would plot 560 distinct points.

There are two limitations to using the color bit to extend the graphics capability of the II Plus. First, color is dependent on location (this does not matter if you have a monochrome display), and second, all the bits of the byte are affected by changing the color bit. In other words, you can't plot points in positions one and two at the same time.

USING THE PROGRAMS

DOUBLE.HIRES

The machine language routine DOUBLE.HIRES (**Listing 28**) is a simple way to generate double Hi-Res graphics. It can be located anywhere in memory and is called from Applesoft with the command:

CALL *D*,*C*,*X*,*Y*

where *D* is the decimal address for the beginning of the machine language routine (768 as listed here); *C* sets the color (0 for black and 1 for white); *X* is the X-coordinate, which ranges from 0-559; and *Y* is the Y-coordinate, which ranges from 0-191. You can use either Hi-Res graphics page with this routine. **Table 6** shows the Applesoft routines that the program uses and their functions.

TABLE 6: Applesoft ROM Routines

Applesoft Routine	Function
CHKCOM	Checks for a comma at the location pointed to by TXTPTR.
TXTPTR	A pointer for the next character or token from a program.
COMBYTE	Checks for a comma and gets a byte in the X-Register (uses TXTPTR).
FRMNUM	Evaluates the expression pointed to by TXTPTR, puts the results into FAC, and makes sure it's a number.
FAC	Applesoft's main floating-point accumulator.
GETADR	Converts FAC into a two-byte integer and stores it in LINNUM.
HFNS	Gets and sets coordinates to be plotted in Hi-Res graphics. The program enters HFNS at $F6BF where it picks up the Y-coordinate.
HPLOT	Plots the point at the coordinate set by HFNS.
COLOR	The memory location in which the Hi-Res color byte is stored.
LINNUM	A two-byte location used in Applesoft as a general 16-bit number location.

THE DEMONSTRATION PROGRAM

The Applesoft demonstration program DHR.DEMO (**Listing 29**) is a collection of shapes drawn in double Hi-Res and standard Hi-Res graphics. The double Hi-Res routine is loaded at decimal location 768. We can see that the screen appears contracted in double Hi-Res mode. A 50-point by 50-point box looks like a rectangle, for instance, and a circle in Hi-Res looks like an oval. (To obtain more normal proportions, double the total range of all X-coordinates in double Hi-Res.) However, resolution is finer — a sine wave has more plotted points and looks smoother.

ENTERING THE PROGRAMS

To key in the program DOUBLE.HIRES, type it in as it is shown in **Listing 28** and save it to disk with the command:

BSAVE DOUBLE.HIRES,A$300,L$2E

To key in the Applesoft demonstration program DHR.DEMO, type it in as shown in **Listing 29** and save it to disk with the command:

SAVE DHR.DEMO

HOW DOUBLE.HIRES WORKS

The program is well-documented. When called, the routine first initializes the color byte to black (zero). It next uses COMBYTE to check the CALL statement for a comma and puts the color into the X-Register. If the color is white, the color byte is changed (set to 127 or color 3).

The routine now needs to get the X-coordinate. CHKCOM is used to check for a comma. FRMNUM then gets the X-coordinate from the CALL statement and GETADR evaluates it and stores it in LINNUM. The Carry bit is then cleared and the X-coordinate divided by two. This is accomplished by rotating the byte in LINNUM to the right using the ROR instruction.

After the rotation, the Carry bit, which now contains the value previously in bit 0, is checked to see if the coordinate is odd or even. If the Carry is set, the coordinate is odd and the sign bit of the color byte is set. If the Carry is clear, the X-coordinate is even and the sign bit of the color byte remains zero. The even coordinate plots the left half of the plotting bit, and the odd coordinate plots the right half. The X-coordinate in LINNUM is now in the range of the Applesoft HPLOT command. The routine enters the Applesoft routine, HFNS, where it gets the Y-coordinate and then HPLOTs to plot the point. Control is then returned to the BASIC program.

LISTING 28: DOUBLE.HIRES

```
;
;**********************
;*    DOUBLE.HIRES    *
;* BY ALGIS MATYCKAS  *
;* COPYRIGHT (C) 1985 *
;* BY MICROSPARC, INC *
;* CONCORD, MA  01742 *
;**********************
;
;MICROSPARC ASSEMBLER SOURCE
;
;--- APPLESOFT ROUTINES ---
;
CHKCOM    EQU $DEBE        ;CHECKS TXTPTR FOR COMMA
```

```
                 COMBYTE  EQU $E74C        ; GET A BYTE IN X REG
                 FRMNUM   EQU $DD67        ;EVALUATE EXPRESSION
                 GETADR   EQU $E752        ;CONVERT INTO INTEGER
                 HFNS2    EQU $F6BF        ; SET COORD.
                 HPLOT    EQU $F457        ;PLOTS A POINT AT COORD SET
                 COLOR    EQU $E4          ;HI-RES COLOR BYTE
                 LINNUM   EQU $50          ;16 BIT NUMBER LOCATION
                 ;
                 ;ROUTINE CAN BE RELOCATED ANYWHERE IN MEMORY
                 ;
                          ORG $300
                 ;
                 ;--- GET COLOR (BLACK OR WHITE) ---
                 ;
0300  A9 00               LDA #$00         ;INITIALIZE COLOR BYTE
0302  85 E4               STA COLOR        ;WITH BLACK
0304  20 4C E7            JSR COMBYTE      ;GET COLOR FROM CALL
0307  E0 00               CPX #$00         ;IS COLOR BLACK?
0309  F0 04               BEQ BLACK        ;THEN GO ON ELSE
030B  A9 7F               LDA #$7F         ;CHANGE COLOR BYTE
030D  85 E4               STA COLOR        ;TO WHITE
                 ;
                 ;
                 ;--- GET PLOTTING COORDINATES ---
                 ;
030F  20 BE DE   BLACK    JSR CHKCOM       ;CHECK FOR COMMA AND
0312  20 67 DD            JSR FRMNUM       ;GET X COORDINATE
0315  20 52 E7            JSR GETADR     ;EVALUATE AND STORE IN LINNUM
0318  18                  CLC              ;CLEAR CARRY REG.
0319  66 51               ROR LINNUM+1     ;DIVIDE BY TWO
031B  66 50               ROR LINNUM       ;AND CHECK IF EVEN
031D  90 08               BCC EVEN         ;IF EVEN THEN PLOT
031F  D8                  CLD          ;ELSE CLEAR FOR BINARY ADDITION
0320  18                  CLC              ;CLEAR CARRY REG.
0321  A5 E4               LDA COLOR        ;LOAD COLOR BYTE
0323  69 80               ADC #$80         ;SET SIGN BIT
0325  85 E4               STA COLOR        ;AND STORE COLOR BYTE
                 ;
                 ;
                 ;--- PLOT POINT ---
                 ;
0327  20 BF F6   EVEN     JSR HFNS2        ;GET Y COORDINATE
032A  20 57 F4            JSR HPLOT        ;PLOT THE POINT
032D  60                  RTS              ;RETURN

000  ERRORS

0300  HEX START OF OBJECT
032D  HEX END OF OBJECT
002E  HEX LENGTH OF OBJECT
95AA  HEX END OF SYMBOLS
```

LISTING 29: DHR.DEMO

```
1  REM ***********************
2  REM *       DHR.DEMO       *
3  REM * BY ALGIS MATYCKAS    *
4  REM * COPYRIGHT (C) 1985 *
5  REM * BY MICROSPARC, INC *
6  REM * CONCORD, MA  01742 *
7  REM ***********************
50  REM  INITIALIZE AND SET UP HGR SCREEN
60  HOME : HGR : HCOLOR= 3: PRINT
70  PRINT  CHR$ (4);"BLOAD DOUBLE.HIRES"
80 DHR = 768: REM  ADDRESS OF    DOUBLE HI-RES ROUTINE
90  HPLOT 140,0 TO 140,159: HPLOT 0,159 TO 279,159
100  INVERSE : VTAB 21: PRINT "   DOUBLE HIRES
     HIRES"; TAB( 40);" "
110  VTAB 22: PRINT  TAB( 40);" ";
120  VTAB 23: PRINT  TAB( 40);" ";: NORMAL
130  REM  **** DRAW BOX ****
140  VTAB 22: HTAB 15: PRINT "50 X 50 BOX"
150  REM  DOUBLE HI-RES
160  FOR X = 115 TO 165: CALL DHR,1,X,55: NEXT
170  FOR Y = 55 TO 105: CALL DHR,1,165,Y: NEXT
180  FOR X = 165 TO 115 STEP  - 1: CALL DHR,1,X,105: NEXT
190  FOR Y = 105 TO 55 STEP  - 1: CALL DHR,1,115,Y: NEXT
200  REM  STANDARD HI-RES
210  FOR X = 185 TO 235: HPLOT X,55: NEXT
220  FOR Y = 55 TO 105: HPLOT 235,Y: NEXT
230  FOR X = 235 TO 185 STEP  - 1: HPLOT X,105: NEXT
240  FOR Y = 105 TO 55 STEP  - 1: HPLOT 185,Y: NEXT
250  GOSUB 630: GOSUB 610: REM   WAIT FOR KEYSTROKE AND
     DIVIDE SCREEN
260  REM  **** DRAW CIRCLE ****
270  VTAB 22: HTAB 12: PRINT "CIRCLE RADIUS 50"
280  REM  DOUBLE HI-RES
290 XC = 140:YC = 80:R = 50:PA = 0:PB = 6.28318:DP = .0174532778
300  FOR P = PA TO PB STEP DP:X = R *  COS (P):Y = R *  SIN
     (P):X = XC + X:Y = Y + YC: CALL DHR,1,X,Y: NEXT
310  REM  STANDARD HI-RES
320  HCOLOR= 3
330 XC = 210:YC = 80:R = 50:PA = 0:PB = 6.28318:DP = .0174532778
340  FOR P = PA TO PB STEP DP:X = R *  COS (P):Y = R *  SIN
     (P):X = XC + X:Y = Y + YC: HPLOT X,Y: NEXT
350  GOSUB 630: GOSUB 610
360  REM  **** DRAW SINE WAVE ****
370  VTAB 22: HTAB 1: INVERSE : PRINT  TAB( 15);" ";: NORMAL :
     PRINT "SINE WAVE";: INVERSE : PRINT  TAB( 40);" ";: NORMAL
380  REM  DOUBLE HI-RES
390  FOR A = 0 TO 278
400 X = (A - 140) / 38:Y =  SIN (X):YP = 96 - (Y * 30): IF YP <
     0 AND YP > 191 THEN 420
410  CALL DHR,1,A,YP
420  NEXT
430  REM  STANDARD HI-RES
```

```
440  FOR A = 140 TO 278
450 X = (A - 210) / 19:Y =  SIN (X):YP = 96 - (Y * 30): IF YP <
     0 AND YP > 191 THEN 470
460  HPLOT A,YP
470  NEXT
480  GOSUB 630: GOSUB 610
490  REM  **** DRAW DIAGONAL ****
500  VTAB 22: INVERSE : PRINT  TAB( 8);" ";: NORMAL : PRINT
     "PARALLEL DIAGONAL LINES";: INVERSE : PRINT  TAB( 40);" ";:
     NORMAL
510  REM  DOUBLE HI-RES
520  FOR X = 0 TO 159: CALL DHR,1,X,X: NEXT
530  REM  STANDARD HI-RES
540  HPLOT 140,0 TO 220,159
550  GOSUB 630
560  REM  INSTRUCTIONS *****
570  TEXT : HOME : PRINT  TAB( 14);"DOUBLE HIRES": VTAB 3: PRINT
     "CALL DHR,C,X,Y": PRINT : PRINT " DHR=DECIMAL LOCATION OF
     DOUBLE HI-RES       ROUTINE"
580  PRINT : PRINT " C=COLOR (0=BLACK,1=WHITE)": PRINT : PRINT "
     X=X COORD. RANGE (0 TO 559)": PRINT : PRINT " Y=Y COORD.
     RANGE (0 TO 191)": VTAB 20: PRINT " END OF DEMO"
590  END
600  REM  SUBROUTINE TO DIVIDE   SCREEN
610  HGR : HCOLOR= 3: HPLOT 140,0 TO 140,159: HPLOT 0,159 TO
     279,159: RETURN
620  REM  SUBROUTINE TO WAIT FOR RETURN TO BE PRESSED
630  VTAB 24: PRINT  TAB( 13);"PRESS <RETURN>";
640 X =  PEEK ( - 16384): IF X < 128 THEN 640
650  POKE  - 16368,0
660  IF X <  > 141 THEN 640
670  VTAB 24: HTAB 1: CALL  - 958
680  RETURN
```

Additional Hi-Res Colors

Tired of being limited to the basic Hi-Res colors? This short Applesoft program gives you additional Hi-Res colors like the pros use.

by Matthew M. Storm

Many adventure games and graphics development systems for the Apple are on the market advertise "21 Hi-Res colors" or "100 Hi-Res colors." However, these are not true, solid colors. They are created by combining two or more colors in a palette, or micro-pattern.

There are many methods to do this. Mine is not the fastest, nor the most versatile, but it is short, effective, and easy to understand. If you have a compiler, you may wish to compile it because it is quite slow.

USING THE PROGRAM

Type RUN and the program (**Listing 30**) will ask if you want to generate additional colors with lines or palettes. Palette drawing is for different shades, and line drawing is for excess colors. If you choose palettes, you must enter the numbers of three colors.

The first number is for the background color, the color on which the rest of the dots will be drawn. The second is the color of the dots in the odd-numbered rasters (horizontal lines). The third is the color of the dots in the even-numbered rasters. For any color that you want to make darker, use a 0 followed with the color number typed twice (such as 0,5,5 for dark orange, and 0,2,2 for dark violet).

You will be prompted to select the width (in pixels) and enter the height (in rasters). After this, you will see a square drawn on the screen in the color and dimensions you specified.

To create additional colors for lines, you only have to enter two color numbers — one for the odd-numbered raster, and one for the even-numbered raster. For example, aqua can be made by typing 1,6.

HOW IT WORKS

Palette drawing is done by first plotting a background. Then the program goes through a loop in which the first dot is plotted in the upper-left corner and the loop is stepped by four (a dot is drawn every four spaces). In an even-numbered row, the first dot is drawn four spaces in. Every dot, as you might have noticed, is two pixels wide because not every color can be plotted in every column.

Line plotting is much easier. When you plot in an area, the color of each line alternates. In an area plotted with colors 1 and 6, three rasters high, the color of the rasters alternates 1,6,1.

LISTING 30: HIRES.COLORS

```
1  REM  **********************
2  REM  *     HIRES.COLORS    *
3  REM  *  BY MATTHEW STORM   *
4  REM  * COPYRIGHT (C) 1983 *
5  REM  * BY MICROSPARC, INC *
6  REM  * CONCORD, MA  01742 *
7  REM  **********************
10  HOME : VTAB 22: PRINT "** COPYRIGHT 1983 BY MICROSPARC, INC.
     **": VTAB 24: PRINT "(P)ALETTES OR (L)INES ";: GET G$: IF
     G$ = "L" THEN 160
20  HOME : VTAB 24: INPUT "ENTER COLORS #1, #2, #3 ";A,B,C
```

```
30  IF A > 7 OR B > 7 OR C > 7 THEN  CALL  - 211: GOTO 20
40  INPUT "HOW WIDE ? ";W
50  INPUT "HOW HIGH ? ";H
60  IF H > 189 THEN 50
70  IF W > 279 THEN 40
80  HGR : POKE  - 16302,0: HCOLOR= A: FOR X = 2 TO H + 2: HPLOT
     0,X TO W,X: NEXT
90  POKE  - 16368,0
100  FOR Y = 2 TO H + 2
110  IF Y / 2 =  INT (Y / 2) THEN  HCOLOR= B: FOR X = 1 TO W
     STEP 4: HPLOT X,Y: HPLOT X + 1,Y: NEXT X: GOTO 140
120  HCOLOR= C: FOR X = 3 TO W - 1 STEP 4: HPLOT X,Y: HPLOT X +
     1,Y: NEXT X
130  IF  PEEK ( - 16384) > 127 THEN 150
140  NEXT Y
150  POKE  - 16301,0: POKE  - 16368,0: RUN
160  HOME : VTAB 24: INPUT "ENTER COLORS #1, #2 ";A,B
170  IF A > 7 OR B > 7 THEN  CALL  - 211: GOTO 160
180  POKE  - 16368,0
190  INPUT "HOW WIDE ? ";W
200  INPUT "HOW HIGH ? ";H
210  HGR : POKE  - 16302,0: POKE  - 16368,0
220  IF W > 279 THEN 190
230  IF H > 189 THEN 200
240  FOR Y = 2 TO H + 2
250  IF Y / 2 =  INT (Y / 2) THEN  HCOLOR= A: HPLOT 0,Y TO W,Y:
     GOTO 280
260  HCOLOR= B: HPLOT 0,Y TO W,Y
270  IF  PEEK ( - 16384) > 127 THEN 150
280  NEXT
290  GOTO 150
```

The Discourager

Prefix your favorite program with this short Applesoft routine to prevent unauthorized access. Impervious to <CTRL>C and <RESET> keys, this password scheme truly discourages prying eyes.

by Mark Allen

The Discourager is just what it sounds like — a deterrent to people who accidentally (or otherwise) look through your personal files or programs. Along with giving you protection, The Discourager allows you to disable the <RESET> key and other normal escape routes.

USING THE DISCOURAGER

To use The Discourager, simply type RUN. Remember that "PASSWORD" in **line 140** can be changed to any word or numbers. The program should be appended to your Hello or other important programs. I suggest that you place it at the very beginning of your programs and replace the END statement in **line 130** with a GOTO statement pointing to the first line of the main program. Be careful not to forget the password.

ENTERING THE PROGRAM

To key in the program, type in **Listing 31** as shown and save it to disk with the command:

SAVE DISCOURAGER

HOW IT WORKS

Line 10 starts the program by clearing the screen and POKEing values into memory that will disable the <RESET> key. The values POKEd into locations 1010 and 1011 make up the address of a third place in memory. This third location (-10906) runs the program in memory. So when you press <RESET>, the program will start over.

The value POKEd into memory location 214 makes all Applesoft commands equal RUN. So you may be able to get out of the program, but anything you type will be executed as though you had typed RUN.

Line 20 sets up the loop for the length of your input. The 13 in **line 20** can be changed to any number up to 255, depending on how long your password is. **Line 30** asks for the password.

Line 40 uses the subroutine at **line 140** to put the cursor in the correct position and also checks to see if <RETURN> has been pressed. If it has, control branches to **line 90**. **Line 50** checks for either a space or a right arrow keypress. If one has been pressed, then it proceeds to **line 70**.

Line 60 prints an "X" in order to hide your password and **line 70** puts a space where the cursor was. The old letters picked up by Z$ are added to the new letter or space in **line 80**. **Line 90** checks to see if the input is equal to the password. If it is, then it goes to **line 120**. **Line 100** tells you that you failed to give the correct password and **line 110** restarts the program to give you another chance.

Line 120 tells you that you gave the correct password and sets everything back to normal. **Line 130** ends the program. **Line 140** is a subroutine that positions the cursor by finding how long the prompt (A$) is and then adding that to the number of characters already typed in.

LISTING 31: DISCOURAGER

```
1  REM   **********************
2  REM   *     DISCOURAGER    *
3  REM   *    BY  MARK ALLEN  *
4  REM   * COPYRIGHT (C) 1984 *
5  REM   * BY MICROSPARC, INC *
6  REM   * CONCORD, MA  01742 *
7  REM   **********************
10  TEXT : HOME : POKE 1011,213: POKE 1012,112: POKE 1010,102:
     POKE 214,128: ONERR  GOTO 150
20  FOR I = 1 TO 13
30  VTAB 22: PRINT "** COPYRIGHT 1984 BY MICROSPARC, INC. **":A$
     = "USER NAME - ": VTAB 12: PRINT A$
40  GOSUB 140: GET Z$: IF Z$ =  CHR$ (13) THEN 90
50  IF Z$ =  CHR$ (32) OR Z$ =  CHR$ (21) THEN 70
60  PRINT  CHR$ (88): GOTO 80
70  PRINT  CHR$ (32): GOTO 80
80 K$ = K$ + Z$: NEXT I
90  IF K$ = "PASSWORD" THEN  GOTO 120
100  HOME : VTAB 12: FLASH : PRINT "ACCESS DENIED": NORMAL
110  FOR X = 1 TO 3000: NEXT : HOME : CLEAR : GOTO 20
120  HOME : VTAB 12: PRINT "ACCESS APPROVED": POKE 1011,157:
     POKE 1012,56: POKE 1010,191: POKE 214,0
130  POKE 216,0: END
140  VTAB 12: HTAB  LEN (A$) + I: RETURN
150  IF  PEEK (222) = 255 THEN  RUN
```

Command Handler

Using DOS 3.3 commands from within an assembly language program is simple once you know how DOS handles them. These short examples will get you started.

by Gary Bond

To find out how assembly language programmers manage to RUN or BRUN disk files from within an assembly language program we need to know a little bit about command handlers. The Apple Disk Operating System (DOS) is a language that allows you to input information from the disk drive to the computer, and to output information from the computer to the disk drive. The routines that handle these input/output (I/O) operations are called by the various commands (BRUN, BLOAD, RUN, SAVE, etc.) in the DOS language.

THE COMMAND HANDLER ENTRY POINT TABLE

The command handler interprets the command with the help of the Command Handler Entry Point Table and calls the appropriate routine. Each routine begins at a fixed address or entry point. To use the routine in an assembly language program, just jump to the address of a particular routine. The addresses are given in the Command Handler Entry Point Table (**Table 7**).

TABLE 7: Command Handler Entry Points

INIT	$A54F
LOAD	$A413
SAVE	$A397
RUN	$A4F0
CHAIN	$A4F0
DELETE	$A263
LOCK	$A271
UNLOCK	$A275
CLOSE	$A2EA
READ	$A51B
EXEC	$A5C6
WRITE	$A510
POSITION	$A5DD
OPEN	$A2A3
APPEND	$A298
RENAME	$A281
CATALOG	$A56E
MON	$A233
NOMON	$A23D
PR#	$A229
IN#	$A22E
MAXFILES	$A251
FP	$A57A
INT	$A59E
BSAVE	$A331
BLOAD	$A35D
BRUN	$A38E
VERIFY	$A27D

CATALOGING

Make sure that DOS is loaded and you have a disk that can be cataloged in the drive. Enter the Monitor by typing CALL -151. Then type in A56EG.

We cataloged the disk by directly calling the routine that handles the catalog operation (see **Table 6**). The G following A56E is a Monitor command to run the code starting at the location preceding the G.

If you want to see what the catalog routine looks like, type A56EL. The L is another Monitor command which lists the code in assembly language form.

THE BRUN ROUTINE

Some of the routines can be used alone, but a few require additional information such as a file name. To illustrate, let's use the BRUN routine.

Get out a blank initialized disk, or one that can be clobbered if you make a mistake, and from the Monitor, enter this short machine language program:

```
800:A2 00 E8 8A 9D D0 07 E0 1A D0 F7 60
```

This will serve as the "B" type file for the BRUN routine called from assembly language. Type 800L and press <RETURN>. The listing should look like this:

```
0800-   A2   00         LDX   #$00
0802-   E8              INX
0803-   8A              TXA
0804-   9D   D0   07    STA   $07D0,X
0807-   E0   1A         CPX   #$1A
0809-   D0   F7         BNE   $0802
080B-   60              RTS
```

For those who already know a little assembly language, the results should be clear. For those who don't, type 800G. You should see the alphabet printed in reverse letters somewhere near the bottom of your screen.

To save the program type:

BSAVE ABC,A$800,L$C

BRUNNING AN ASSEMBLY LANGUAGE FILE

To BRUN a file from assemble language requires two things. First, store the file name in a place where DOS will find it. That place is called the primary file name buffer and begins at memory location $AA75. Second, perform either a jump command (JMP) or a jump to subroutine (JSR) followed by the entry point address.

Enter the following machine language program:

```
300:A2 00 A9 A0 9D 75 AA E8 E0 1E D0 F6 A9 C1 8D 75 AA A9 C2 8D
76 AA A9 C3 8D 77 AA 4C 8E A3
```

Now type 300L and study the disassembled listing below:

```
0300-   A2   00         LDX   #$00
0302-   A9   A0         LDA   #$A0
0304-   9D   75   AA    STA   $AA75,X
0307-   E8              INX
0308-   E0   1E         CPX   #$1E
030A-   D0   F6         BNE   $302
```

```
030C-  A9  C1      LDA  #$C1
030E-  8D  75  AA  STA  $AA75
0311-  A9  C2      LDA  #$C2
0313-  8D  76  AA  STA  $AA76
0316-  A9  C3      LDA  #$C3
0318-  8D  77  AA  STA  $AA77
031B-  4C  8E  A3  JMP  $A38E
```

Before trying it out, save the new program with the command:

BSAVE DEMO,A$300,L$1E

After you have both files saved on disk, try running the new program first from the Monitor by entering 300G, and then as a direct disk command by entering BRUN DEMO. The disk file DEMO clears the primary file name buffer, loads the file name to BRUN (ABC), and then jumps to the entry point for the BRUN routine.

Lines 300-30A clear the primary file name buffer by storing A0 (the value for space) in locations $AA75-$AA92. One of the biggest mistakes the beginning programmer makes is not clearing the buffer. The buffer must be clear before you use it because remnants from a larger file name may remain.

Lines 30C-30E store the value C1 in the first memory location of the buffer ($AA75), and the remainder of the program stores the values C2 and C3 in the second and third buffer locations (AA76-AA77). The values C1, C2 and C3 are the hexadecimal representations of the letters A, B and C — which happen to be the file name for the binary file we previously saved to disk. Finally, **line 31B** jumps to the command handler entry point for BRUN.

Try experimenting with the DEMO program by modifying 31B in the different ways shown below. Use the G command to run the new version each time.

31B:4C 71 A2 (will LOCK the ABC files)
31B:4C 75 A2 (will UNLOCK the ABC file)
31B:4C 5D A3 (will BLOAD the ABC file)
31B:4C 7D A2 (will VERIFY the ABC file)
31B:4C 63 A2 (will DELETE the ABC file)

The same rules apply for Applesoft files using the LOAD, RUN and SAVE commands.

DISABLING A DOS COMMAND

It's easy to disable any of the DOS commands. Simply store the value 60 at the Command Handler Entry Point Table address of the command you want to disable. The value 60 is an assembly language command for return from subroutine (RTS). For example, to disable the CATALOG command, enter A56E:60 from the Monitor and try to catalog the disk.

FID Plus

Streamline the FID utility from your DOS 3.3 System Master with these simple enhancements. Eliminate unnecessary keypresses, give the commands mnemonic symbols and change the wildcard character with just a few modifications of the code.

by Joe Humphrey

FID (FIle Developer) is probably the most useful and well-written program that comes with DOS 3.3. However, it has several features which can become quite annoying after a while.

THE FEATURES

1. It uses digits 0-9 to represent commands, rather than using more meaningful mnemonics (such as the letter C to obtain a CATALOG).
2. It requires that you press <RETURN> after every command, even though most commands are only one character (a pet peeve of mine).
3. Instead of using an asterisk (*) as a standard for a wildcard character (such as in "*.SOURCE", which represents all files that end in ".SOURCE"), FID uses the equal sign (=).
4. FID requires that you reenter the slot and drive numbers every time you switch between COPY FILES and other commands, even though they usually keep the same values.

HOW TO FIX THEM

FID starts at memory location $0803 and is immediately followed by data. Therefore, any changes need to be inserted into the program itself. Fortunately all the above features except for number 4 can be fixed despite this restriction, and number 4 isn't too great an annoyance once number 2 has been taken fixed.

To implement the enhancements, you need to do the following:

1.
```
CALL 151
BLOAD FID
```
Enter the monitor, and load FID.

2.
```
1885
<RETURN>
```
Check the byte at 1885. If the value is 00, you should begin entering code at 1886 instead of 1885 as shown in the next step

```
1885:CD CF
 D6 C5
```
Change the COPY FILES command to MOVE FILES, since the COPY command conflicts with the CATALOG command.

```
13AF:CD C3
 D3 D5 CC
 C4 D2 D6
 D1 00 D2
 D1 00 C4
 C3 CC D3
 D5 D6 00
 CD 00 C3
 D2 D3 D1
 00
```
Change the commands from the digits 1-9 to the letters M(ove), C(atalog), S(pace), U(nlock), L(ock), D(elete), R(eset), V)erify and Q(uit).

3. Change some routines so that now, whenever a single character of input is wanted, only the character needs to be typed.

```
08C7:0C FD
 20 ED FD
0AB1:4C C1
 FB A2 0B
 2C A2 0C
 20 CD 0A 20
 0C FD 8D
 00 02 AA 20
 ED FD 20
 8E FD 8A
 A2 01 60
```

Change all references to the routines.

```
0941:BC 0A
0965:BC 0A
098D:BC 0A
09B1:BC 0A
0A73:BC 0A
0B46:B4 0A
0B6B:B4 0A
0C1E:BC 0A
0E72:B7 0A
0E80:B4 0A
0E8B:B7 0A
0FA5:BC 0A
```

4. Change the wildcard character from "=" to "*".

```
0A38:AA
0A50:AA
0CE1:AA
0D00:AA
0D29:AA
```

5. Save the result to a new file.

```
BSAVE FID+,A$0803,L$124E
```

or

Update FID itself.

```
UNLOCK FID
BSAVE FID,A$0803,L$124E
LOCK FID
```

These changes can be made independently of each other so that you can make only those changes, you want.

Label Printer

Mailing list programs are fine if you want to generate hundreds of labels, but what if you want just one or two? This short Applesoft program is designed for these small jobs and gives you a choice of type styles.

by Robert C. Brock

Conventional mailing label programs provide the advantage of access to extensive databases, but these heavy-duty systems rarely lend themselves to generating single labels. Label Printer makes it convenient to print single or multiple labels for envelopes, file folders, disks, etc. It's easy to run, lets you vary the type style and gives you the option of making corrections before the labels are actually printed.

Label Printer was developed for use with the Epson MX-80 printer. As written, the program will let you select standard (10 characters per inch), condensed (16.5 cpi), or double-width (9 cpi) type styles, with line lengths of 25, 45 and 21 characters, respectively. It uses 3 by 7/8 inch labels.

USING LABEL PRINTER

You are first prompted to enter the number of labels to be printed and the number of lines each label is to have. Next, you select the type style. The program prints the line number, and the line length is displayed on the screen with left and right brackets and periods for character spaces. The line length varies with the type style chosen. Enter each line of the label. Then the label is displayed on the screen and you have the option to correct the data. When everything is correct as displayed, the printer is activated and the label(s) is printed. You can run more labels or terminate the program with a single keystroke.

ENTERING THE PROGRAM

To key in the program, enter **Listing 32** as shown and save it to disk with the command:

```
SAVE LABEL.PRINTER
```

HOW THE PROGRAM WORKS

Throughout the program, input is checked to reject out of range entries where applicable. **Lines 160-180** determine the type style and line length. Epsons, as well as other printers, use escape sequences to vary type styles. These consist of <ESC> (CHR$ (27)) followed by another character which changes the printer's mode. Variable LT sets the style and variable DW holds the value for the double width mode. Variable SP defines the line length. If you don't have an Epson, your printer's commands can be substituted here.

Lines 190-230 make up the input routine. The looping process is determined by the variable N which specifies the number of lines in the label. **Lines 240-260** display the label before printing and offer the opportunity to make corrections. When everything is correct, the processing moves to **lines 290 and 300** where the printer is turned on and the label printing routine takes over.

CUSTOMIZATION

Label size and type style are a matter of need and personal preference. By changing the line length (SP) and type style (LT and DW), different label sizes and varieties can be developed.

LISTING 32: LABEL.PRINTER

```
10  REM   **********************
20  REM   *   LABEL.PRINTER     *
30  REM   * BY ROBERT C. BROCK *
40  REM   * COPYRIGHT (C) 1984 *
50  REM   * BY MICROSPARC, INC *
60  REM   * CONCORD, MA  01742 *
70  REM   **********************
80  TEXT : HOME : CLEAR : POKE 34,10: DIM L$(5)
90  INVERSE : PRINT "***   L A B E L   P R I N T E R  ***":
     PRINT "***      BY  ROBERT C. BROCK         ***": PRINT "*
     COPYRIGHT 1984 BY MICROSPARC, INC *": NORMAL
100  VTAB 5: INPUT "HOW MANY LABELS TO PRINT?     ";NN
110  VTAB 6: INPUT "HOW MANY LINES PER LABEL?     ";N
120  IF N < = 0 OR N > 5 THEN  GOTO 110
130  VTAB 8: PRINT "TYPE STYLE:  1) STANDARD": PRINT  TAB(
     14)"2) CONDENSED": PRINT  TAB( 14)"3) DOUBLE WIDTH  ";:
     INPUT "";TY$
140  IF TY$ = "1" OR TY$ = "2" OR TY$ = "3" GOTO 160
150  GOTO 130
160  IF TY$ = "1" THEN LT = 18:SP = 25:DW = 18
170  IF TY$ = "2" THEN LT = 15:SP = 45:DW = 18
180  IF TY$ = "3" THEN LT = 14:SP = 21:DW = 15
190  FOR NL = 1 TO N
200  VTAB 12: PRINT : PRINT "LINE #";NL;":": PRINT : PRINT  CHR$
     (91);: FOR L = 1 TO SP: PRINT ".";: NEXT L: PRINT  CHR$
     (93)
210  VTAB 15: HTAB 2: INPUT "";L$(NL)
220  IF  LEN (L$(NL)) > SP THEN  HOME : GOTO 200
230  NEXT NL
240  HOME : VTAB 14: PRINT "THIS IS THE WAY THE LABEL WILL
     LOOK:": PRINT
250  FOR C = 1 TO 5: PRINT L$(C): NEXT C
260  VTAB 22: PRINT "IS THIS CORRECT? (Y/N) ";: GET AN$
270  IF AN$ = "Y" THEN  POKE 34,3: GOTO 290
280  IF AN$ = "N" THEN  HOME : GOTO 190
290  HOME : PRINT : PRINT  CHR$ (4)"PR#1"
300  FOR C = 1 TO NN: FOR I = 1 TO 5: PRINT  CHR$ (DW); CHR$
     (LT);L$(I): NEXT I: PRINT : NEXT C
310  PRINT  CHR$ (4)"PR#0": POKE 34,3: HOME : VTAB 5: PRINT "RUN
     MORE LABELS? (Y/N) ";: GET AN$: PRINT AN$
320  IF AN$ = "Y" THEN  GOTO 80
330  POKE 34,0: HOME : END
```

Break Processor

Use this handy technique to insert break points in your assembly language programs. Examine the processor registers before continuing the execution of your program.

by John J. Broderick

When writing assembly language code, it is often useful to be able to stop, display the registers, examine memory, and then continue processing. However, your Apple may not have enough memory to use a large debugging program and many debugging programs cannot debug past DOS. In these cases, the subroutine below could help:

```
BREAK
            BRK
            NOP
            PLA
            PLA
            JSR $FF3F
            RTS
```

Wherever you want to stop your program, place a JSR BREAK directed to this subroutine. The processor will recognize the BRK instruction and perform a system break displaying the registers, Processor Status flags and stack. To continue, press G and <RETURN>.

HOW IT WORKS

At a BRK instruction, the Apple ignores the next byte (a NOP) and places the address of the first PLA in $3A and $3B. It also stores the registers in page 0 from memory $45-$49.

Pressing the G and <RETURN> keys causes the Apple to get the address from memory locations $3A and $3B and begin executing the instruction at that address. The PLAs are necessary to discard the two bytes that were pushed on the stack by the G<RETURN>.

The next two bytes on the stack will be used by the following RTS, returning to your program. The JSR $FF3F restores the proper contents of the registers before continuing.

ADD A ONE TO THE A-REGISTER

Listing 33 is a little program that keeps adding a one to the A-Register. Please refer to Appendix A for help in entering this program. If you key it in from the Monitor, save it to disk with the command:

BSAVE BREAK.PROCESSOR,A$5000,L$17

After you assemble this program into memory, turn on the printer and begin executing these instructions at 5000 hex: Type CALL -151 to get into the Monitor . Then type 36:0 C1 (to turn on printer) and 5000G (to begin executing).

Figure 6 shows a sample printout. The A-Register starts by containing zero; however, a one is added to it just before the break. All other registers should remain the same.

Figure 6: Sample Printout

```
5011- A=01 X=02 Y=03 P=30 S=EE
*G
5011- A=02 X=02 Y=03 P=30 S=EE
*G
5011- A=03 X=02 Y=03 P=30 S=EE
```
etc.

If you want to place the BREAK subroutine directly into memory at 6000 hex, enter from the Monitor:

```
6000:00 EA 68 68 20 3F FF 60
```

You would then JSR $6000 to execute a break.

LISTING 33: BREAK.PROCESSOR

```
                          ;
                          ;
                          ;         BREAK.PROCESSOR
                          ;
                                    ORG $5000
 5000   A9 00     INIT      LDA #0          ;(LOAD THE REGS WITH ANYTHING)
 5002   A2 02               LDX #2
 5004   A0 03               LDY #3
 5006   18                  CLC            ;ALWAYS CLEAR THE CARRY BEFORE G
                          ;
 5007   69 01     LOOP      ADC #1          ;(ADD 1 TO THE A REGISTER)
 5009   20 0F 50            JSR BREAK
 500C   4C 07 50            JMP LOOP        ;(NEVER ENDING LOOP FOR TEST)
                          ;
 500F   00        BREAK     BRK
 5010   EA                  NOP
 5011   68                  PLA
 5012   68                  PLA
 5013   20 3F FF            JSR $FF3F
 5016   60                  RTS

000  ERRORS

5000  HEX START OF OBJECT
5016  HEX END OF OBJECT
0017  HEX LENGTH OF OBJECT
95E9  HEX END OF SYMBOLS
```

Decision Maker

Use this short program to help you with the decisions in your life. Just enter your options and rate pairs of choices to determine your priorities.

by Beirne L. Konarski

Major decisions are seldom easy to make. There are too many factors to consider and weigh, one against the other. For instance, when I considered a career change, I thought about factors like location, type of industry, degree of independence and the required skills. When you buy a car, you must decide on your priorities: do you want a car with great gas mileage, power, comfort, style or are you just looking for the cheapest way to get around? When you purchased your computer, you probably went through a process of setting priorities and comparing memory size, portability, expandability, and available languages and software.

Decision Maker is a short program can help you make those complicated decisions by allowing you to choose among different options. Then, based on your preferences, it lists the options in order of their priority and displays the corresponding weighting factors. The program uses a method of bubble sorting that even the inexperienced programmer can understand.

USING DECISION MAKER

The program first asks how many options are to be evaluated. You are then prompted to enter each option. Next, pairs of options are displayed on the screen. Enter a 1 or a 2 for each set to indicate your preference. At times the decision may be difficult, but usually one choice is preferable. After the pairs are compared, the computer displays the options, with the most favored at the top and the number of favored comparisons to the right.

ENTERING THE PROGRAM

To key in the Decision Maker, enter the Applesoft program shown in **Listing 34** and save it on disk with the command:

SAVE DECISION.MAKER

HOW THE PROGRAM WORKS

Program execution begins with the prompt to obtain the number of options to be compared. This number is used as a loop index to obtain the list of options and to store them in the array LIS$() **(lines 170-200)**. The array structure allows easy comparisons between pairs of choices in the nested loops in lines **210-330**. The outer loop traverses the list from the first element to the next-to-last element, using the inner loop to compare each successive array element with every element that follows it. In this way, each item is compared to every other item without duplication.

Before the results can be displayed, the array elements must be sorted. I chose a simple bubble sort for this purpose. If the bubble sort is unfamiliar, trace through a sample array on a piece of paper and you will find that each time through the loop, the highest value is pushed to the top. Finally, a simple loop prints the array and you are asked if you want to restart the program.

LISTING 34: DECISION.MAKER

```
10  REM *************************
20  REM *    DECISION.MAKER     *
30  REM * BY BEIRNE L. KONARSKI *
40  REM * COPYRIGHT (C) 1985    *
50  REM * BY MICROSPARC, INC.   *
60  REM * CONCORD, MA  01742    *
70  REM *************************
80  REM
90  REM  *********************
100  REM   CHOOSE PRIORITIES
110  REM  *********************
120  HOME
130  VTAB 5: HTAB 14: PRINT "DECISION MAKER"
140  HTAB 19: PRINT "BY"
150  HTAB 12: PRINT "BEIRNE L. KONARSKI": PRINT : PRINT "*
     COPYRIGHT (C) 1985 MICROSPARC, INC. *"
160  VTAB 13: HTAB 1: CALL  - 958: INPUT "HOW MANY CHOICES DO
     YOU HAVE? ";C$:C =  VAL (C$): IF C < 3 THEN 160
170  DIM LIS$(C),TALLY(C)
180  FOR X = 1 TO C
190  PRINT "CHOICE #"X;: INPUT LIS$(X)
200  NEXT
210  FOR X = 1 TO C - 1
220  FOR Y = X + 1 TO C
230  HOME
240  PRINT "1. "LIS$(X)
250  PRINT
260  PRINT "2. "LIS$(Y)
270  PRINT
280  PRINT "WHICH IS MORE IMPORTANT TO YOU? ": GET ANSWER$
290  IF ANSWER$ = "1" THEN TALLY(X) = TALLY(X) + 1: GOTO 320
300  IF ANSWER$ <  > "2" THEN  PRINT : PRINT "1 OR 2 PLEASE":
     PRINT : PRINT : PRINT "PRESS <RETURN> TO CONTINUE": GET A$:
     GOTO 230
310 TALLY(Y) = TALLY(Y) + 1
320  NEXT
330  NEXT
340  REM   ********************
350  REM   DISPLAY PRIORITIES
360  REM   ********************
370  HOME
380  FOR X = C - 1 TO 1 STEP  - 1
390  FOR Y = 1 TO X
400  IF TALLY(Y) < TALLY(Y + 1) THEN HOLD = TALLY(Y):TALLY(Y) =
     TALLY(Y + 1):TALLY(Y + 1) = HOLD:TEMP$ = LIS$(Y):LIS$(Y) =
     LIS$(Y + 1):LIS$(Y + 1) = TEMP$
410  NEXT : NEXT
420  PRINT "HERE IS THE ORDER OF PRIORITY"
430  FOR X = 1 TO C: PRINT LIS$(X);:HTAB 20: PRINT TALLY(X):NEXT
440  VTAB 22: PRINT "WOULD YOU LIKE TO RESTART? (Y/N)";: GET A$
450  IF A$ = "Y" THEN  CLEAR : GOTO 120
460  PRINT : HOME : END
```

Print Using TAB

Overcome Applesoft's weakness in formatting output with these short Applesoft routines. Align decimal points or format dollar amounts without resorting to machine language.

by Clay Carr

One of the features that Applesoft doesn't have is a decimal tabulation function. This function allows you to print columns of numbers so that the decimal points are lined up neatly. To see just how far away Applesoft is from this, run the following:

```
10 FOR J = 400 TO -400 STEP -100: PRINT J/7: NEXT: END
```

Obviously, this won't do for any but the roughest program outputs.

The most common way to deal with this problem is to create a PRINT USING function. This is either a subroutine or a machine language program that can be called by an Applesoft program. (A very complete machine language version of this is presented in C. Bongers' article "Amper Print-Use Program" in *Nibble Express* Vol. II.)

For my purposes, though, this is overkill. I seldom need to handle anything complex enough to require more than a few decimal places (including negative numbers) and a dollar sign. As a result, I have found some simpler alternatives for formatting real number fields. In these examples, I will be making changes in lines. Line numbers in the text refer to the ones most recently changed.

USING BOOLEAN LOGIC

The simplest and fastest approach is to use the Boolean capabilities within Applesoft. (See Don Ravey's article How to Use Boolean in *Nibble* Vol. 3/No. 5.) This approach is based on the structure of logical comparisons in Applesoft. If the result of a comparison is true, a one is returned; if the result is false, a zero is returned. Using only this structure, it's easy to create a subroutine which produces a true decimal tab. The basic form of the subroutine is:

```
1000 T = 10 - (N>=10000) - (N>=1000) - (N>=100) - (N>=10) -
     (N>=0)
1010 RETURN
```

To test the results, let's modify the simple number-generation routine above:

```
10 FOR J = 400 TO -400 STEP -100: N=J/7: GOSUB 1000: PRINT
     TAB(T)N: NEXT: END
```

When you run the routine, you'll see all of the positive numbers neatly lined up on their decimal points. The value for T can be any number, depending on where you want the number to print. But although the positive numbers look great, the negative numbers are not correctly lined up. Let's use Boolean logic to correct this with a few simple changes:

```
1000 I = 1 - (2*(N<0))
1010 T = 10 - ((N*I) >= 10000) - ((N*I) >= 1000) - ((N*I)>=100)
     - ((N*I)>=10) - ((N*I)>=0) - (N<0)
1020 RETURN
```

What have we done? The statement I = 1 - (2*(N<0)) sets I equal to -1 if N is negative. Now, whenever the Boolean comparisons are made, any negative numbers are turned into positive ones. (The same thing could also be done using the ABS function.) A final space is subtracted if the number is negative (N<0), to create a slot for the negative sign.

MAKE ROUNDING OFF A SNAP

This is now a full-fledged decimal tab function, as you will see if you run **line 10** again. But most of the time we don't need all of those decimal places. There's a neat trick with the INT function that makes rounding off to any place a snap. Let's make it the first statement of the subroutine:

```
1000 N = INT (N*100+.5)/100
1010 I = 1 - (2*(N<0))
1020 T = 10 - ((N*I) >= 10000) - ((N*I) >= 1000) - ((N*I) >=
     100) - ((N*I) >= 10) - ((N*I) >= 0) - (N<0)
1030 RETURN
```

Line 1000 rounds off N to two decimal places. The number of places to round off is simple to select. It is the inverse of the multiplier used inside the INT function. In the example above, the multiplier is 100, so the number will be rounded off to 1/100, or two decimal places. (If the division by 100 were omitted, the function would convert the decimal fraction to a percentage. Any decimal fraction can be converted to a percentage if the divisor outside the INT function is 1/100th of the value of the multiplier inside the function.)

DEALING WITH DOLLAR FIELDS

The subroutine now has a complete decimal tab function that includes rounding off to any decimal place as you can see by rerunning **line 10**. If you just need to print numbers, it is short and fast. It will also work if you need to print dollar amounts for positive numbers; just put a dollar sign ($) immediately after TAB(T). But the subroutine has three drawbacks when used for dollar fields:

1. If the number is negative, the negative sign prints after the dollar sign ($-12.5).

2. If the number is less than one, no leading zero is printed before the decimal (which is not necessary, but produces a more professional appearance).

3. If there is only one decimal place used ($37.5), it will print without a trailing zero. Also, the decimal point will be omitted if the number is an integer.

I have not found a way to solve these problems with a decimal tab subroutine. Instead, we need to construct a completely formatted result (often called a PRINT USING function, since that is the command used to format numbers in many BASICs). The essential difference between decimal tabulation and formatting is that in the latter, the number is converted into an alpha field.

We could begin with the rounding function in **line 1000** and convert the rounded number to a string:

```
1000 N = INT (N*100+.5)/100
1010 N$ = STR$(N)
1020 RETURN
```

This is straightforward, but it gets us into trouble right away. To see the problem, modify the number generator line slightly and run it. Since we're not computing a tab setting, drop the tab and substitute N$ for N:

```
10 FOR J = 400 TO -400 STEP -100: N = J/7: GOSUB 1000: PRINT N$:
     NEXT: END
```

Merely changing the number (N) to a string (N$) hasn't helped to keep trailing zeros in the cents field: 37.5 remains 37.5, and 25 remains 25. Fortunately, a sneaky bit of addition and subtraction solves this:

```
1000 N = INT (N*100+.5)/100+.001
1010 N$ = STR$(N)
1020 N$ = "$" + LEFT$ (N$,LEN(N$) - 1)
1030 RETURN
```

When you run line 10, you'll find that the trailing zeros have been captured. We added .001 to the initial number, guaranteeing that the string will pick up the decimal places. Then we simply drop the rightmost character and voila — there are two decimal places in every situation!

We solve the problems of negative numbers and zeros before decimal fractions with two new leading lines:

```
1000 N$ = "$": IF ABS (N) < 1 THEN N$ = N$ + "0": IF N =0 THEN
     N$ = "$0.00": GOTO 1050
1010 IF N < 0 THEN N$ = "-" + N$
1020 N = INT (ABS(N)*100+.5)/100 + .001
1030 N$ = N$ + STR$(N)
1040 N$ = LEFT$ (N$,LEN(N$) - 1)
1050 RETURN
```

Line 1000 adds a leading zero to each decimal fraction. It also sets N$ equal to "$0.00" if the number is zero; without this instruction, Applesoft has the disconcerting habit of printing zero in the exponential format. **Line 1010** takes care of negative numbers by providing a negative sign in front of the dollar sign.

We need to one more line to add leading spaces and then a final line for RETURN:

```
1000 N$ = "$": IF ABS(N) < 1 THEN N$ = N$ + "0": IF N = 0 THEN
     N$ = "$0.00": GOTO 1050
1010 IF N<0 THEN N$ = "-" + N$
1020 N = INT (ABS(N) * 100+.5)/100 + .001
1030 N$ = N$ + STR$(N)
1040 N$ = LEFT$ (N$,LEN(N$) - 1)
1050 IF LEN (N$) < 10 THEN N$ = " " + N$: GOTO 1050
1060 RETURN
```

This gives us the complete formatting subroutine for dollar fields, complete with leading blanks, which can be verified by running **line 10**.

SUMMARY

The first routine will work satisfactorily (and rapidly) with any numbers that don't require prefixes (such as dollar signs) and where trailing zeros aren't important. The second one, which is slightly more complex and takes a bit longer to execute, will handle any real number and can be used with or without prefixes. Both are short, with almost unnoticeable execution times.

FOR THOSE WHO WOULD LIKE PERFECTION

Actually, there is a flaw in the round-off formula in both subroutines. When I first submitted this article, *Nibble* editors wrote back and asked me to put the two subroutines on disk. I did, and added a short subroutine so that they could enter any value to check out both of the subroutines. Since I also wanted to be sure they worked, I entered a variety of numbers. A glitch popped up. Mind you, the glitch shouldn't exist.

According to my very limited mathematical knowledge, the rounding formulas above should work 100% of the time. But they don't. If you enter 1.115 or -1.115 into either, you get 1.11 and -1.11. The same thing happens with 2.225 and 3.335. It doesn't happen with 4.445 or higher repeats.

Fortunately, there is a simple fix. In both subroutines, .5 is added to the number in the parentheses to ensure that rounding off occurs. All you need to do is to add .51 instead. Don't ask me why it works — I assume that it has to do with the way Applesoft translates hexadecimal values into decimal values.

Applesoft Variable Dump

Checking the value of a variable is one of the most powerful methods for debugging a program. This Applesoft utility will display the values of almost all of your variables and can be called from within the program any time.

by Tom Gabriele

Back in the days when a computer's memory was made up of magnetic doughnuts called core, a major tool used to find bugs in programs was the core dump. A core (or memory) dump is the output (usually on a printer) of the contents of all the memory locations in a specified range of addresses. By dumping the portion of memory where program variables are stored, the programmer gets a "snapshot" of the state of the program when the memory dump was done.

Memory dumps are usually printed in hexadecimal format, so it's generally a rather formidable task to decipher the variable values. Apple memory dumps can be obtained through use of the Monitor, but the dump is still in the relatively inscrutable hex code.

When debugging an Applesoft program, you can look at the value of any variable or set of variables by simply PRINTing them. These diagnostic PRINT statements can be put into the program itself, or you can execute them in the immediate mode during a break in program execution.

One difficulty with this practice is that in many cases you do not know which variable (or combination of variables) has the value that is causing the program to malfunction. The Variable Dump program is a simple way to dump the values of all program variables to give you a complete snapshot of the state of the program.

Variable Dump (**Listing 35**) handles all types of variables and single dimensional arrays. It does not handle multidimensional arrays. If it did, not only would the program be overly complex and long, but the dump of multiple multidimensional arrays would be voluminous and not generally helpful in debugging. You can selectively dump desired elements of such multidimensional arrays through the conventional PRINT statement technique.

Variable Dump identifies each variable by only the first two letters in its name. This should not be regarded as a limitation because Applesoft itself only uses the first two letters to identify variable names. Thus, if you are uncertain which program variable is being dumped, this ambiguity is caused by two or more variable names beginning with the same two letters, which Applesoft treats as a single variable.

USING VARIABLE DUMP

The program can be used in three simple steps.

1. Add the Variable Dump program statements to the program being tested.
2. Run the program being tested.
3. Transfer control to the variable dump routine.

Adding Variable Dump to Your Programs

Getting the dump statements into the program under test can be accomplished in several ways. Of course, its statements could be typed directly into your program. However, this is tedious and time-consuming, and would discourage frequent use of the routine. A more efficient method is to EXEC the statements from a text file into the program under test..

You can create an EXEC file by adding the following line to **Listing 35**:

```
1 PRINT CHR$(4)"OPEN VARDUMP": PRINT CHR$(4)"WRITE VARDUMP":
    POKE 33,33:LIST 63000,: PRINT CHR$(4)"CLOSE"
```

Thereafter, you need only enter the command EXEC VARDUMP whenever you want to add the variable dumping routine to your program.

An alternative for those who have the Apple DOS 3.3 Tool Kit is to use the APA utility to:

1. Hide in memory the program under test.
2. Load the variable dump program statements.
3. Merge the program under test.

You can accomplish the same task with MicroSPARC's GALE program.

Running the Program Under Test

The second step is to run the program under test. Its execution could be interrupted at appropriate places to display variable dumps. For example, dumps could be displayed just before error symptoms appear, after they show up, when the program bombs, at the end of each iteration through a long loop of statements, or before and after critical and complex computations. The important point is that the program under test must be run in combination with the variable dump program.

Transferring Control

The third step is to get the Apple started on **line 63000**, the beginning of the variable dump routine. This can be done in a number of ways. The simplest way is, after stopping the program, give the immediate command GOTO 63000. The ONERR GOTO 63000 statement could also be inserted early in the program to automatically display a dump whenever an Applesoft or DOS error occurs.

Another very effective way of invoking variable dumps is to sprinkle GOSUB 63000 statements at appropriate places in the program. The resulting series of program state snapshots can provide an informative, dynamic description of program behavior. This debugging technique, combined with the Applesoft TRACE facility, should trap even the most elusive bug. Of course, to use this GOSUB approach, a RETURN must replace the END statement in **line 63140**.

A printed record of variable values can be obtained in the normal manner by enabling the printer with a PR#*n* statement (where *n* is the printer slot number) prior to executing the dump routine.

LIMITATIONS

The variable dump statements are numbered beginning with **line 63000**. Therefore, programs with extremely high statement numbers cannot be tested. Rarely, however, do programs have statement numbers in this range.

Since Variable Dump is written in Applesoft it has its own variables. Programs that use these same variable names cannot be tested. To minimize the chance of this, the dump program contains only 10 variables named Z0-Z9. For those very few programs that may have one or two such variables, the conflicting dump routine variables could be changed to Integer type variables, namely Z0%-Z9% (except for Z3).

HOW IT WORKS

By studying the 45 statements that make up the dump routine, one who is unfamiliar with how the Apple stores numerical and string values can gain a good working knowledge of those Data Table formats. These formats are diagrammed on p. 137 of the *Applesoft Manual* (p. 217 of the *Applesoft BASIC Programmer's Reference Manual for the IIe*). The pointers into these tables at memory locations $69-$6E are defined on p. 140 of that manual (p. 278 in the IIe manual).

The operation of the dump program is rather simple. The utility is modularized into a number of subroutines to make it easier to understand and to minimize the number of statements. It first scans through all the simple variables and then all the arrays. Variable Z9 holds the address of the next variable to be dumped, while Z8 marks the end of that table of variables. The first statement ensures that entries are made in the variable table for all the variables in the dump program itself. This guarantees that the format and pointers of those data tables will be stable (fixed) while the dump utility scans them.

As the scan reaches each variable, the first two characters of its name are saved in Z1 and Z2 while it is temporarily renamed Z3. The value of Z3 is then displayed on the screen following the name contained in Z1 and Z2. After the deed is done, the correct name is restored (POKEd) back to the variable, and the scan continues to the next variable.

As indicated in the *Applesoft Manual*, variable type is indicated by the most significant bits (MSB) of the ASCII characters coding the first two letters of the variable name. If both MSBs are zero, the variable is real. If both are one, the variable is integer. If the first character MSB is a zero but the second character MSB is a one, then the variable is a string. (The older manual seems to have this one backwards.)

The dump utility stores these MSBs in variables Z4 and Z5. It uses them to decide the type of the variable being scanned so that it can use the appropriate variable type for Z3 when it prints its value.

In scanning the simple variables, the Z9 pointer is incremented by seven to point to the next variable. (Seven bytes are used to describe each simple variable.) In scanning the array variables, the Z9 pointer is incremented by the total number of bytes used to store that array. This byte count is contained in the third and fourth bytes of that array's Data Table.

After the name of each simple variable is extracted from the table, it is checked to see if it is a variable of the utility program itself (Z0-Z9), since these are stored in the same Data Table along with the variables of the program under test. If the variable being scanned is a dump program variable, it is simply skipped. The number of dimensions of each array variable (in the fifth byte of its Data Table) is also checked. If it is greater than one, that variable is skipped.

LISTING 35: VARIABLE.DUMP

```
10  REM    *********8.9.84*******
20  REM    *     VARIABLE.DUMP    *
30  REM    *  BY  TOM GABRIELE  *
40  REM    * COPYRIGHT (C) 1984 *
50  REM    * BY MICROSPARC, INC *
60  REM    * CONCORD, MA  01742 *
70  REM    **********************
63000 Z0 = 0:Z1 = 0:Z2 = 0:Z4 = 0:Z5 = 0:Z6 = 0:Z7 = 0:Z8 = 0:Z9
     = 0
63010  GOSUB 63360
63020  HOME : PRINT "** COPYRIGHT 1984 BY MICROSPARC, INC. **":
     PRINT : PRINT "APPLESOFT VARIABLE DUMP": PRINT
63030  IF Z9 = Z8 GOTO 63130
63040  GOSUB 63270
63050  GOSUB 63320: IF (Z0) GOTO 63120
63060  GOSUB 63300
63070  PRINT  CHR$ (Z1); CHR$ (Z2);
63080  IF (Z4) THEN  PRINT "% = ";Z3%: GOTO 63110
63090  IF (Z5) THEN  PRINT "$ = ";Z3$: GOTO 63110
63100  PRINT " = ";Z3
63110  GOSUB 63310
63120 Z9 = Z9 + 7: GOTO 63030
63130  GOSUB 63440
```

```
63140  IF Z9 >  = Z8 THEN  PRINT : PRINT "END OF VARIABLE DUMP":
    END
63150  GOSUB 63270
63160  PRINT "ARRAY ";: GOSUB 63400
63165  IF  PEEK (Z9 + 4) <  > 1 THEN  PRINT " HAS "; PEEK (Z9 +
    4);" DIMENSIONS.": GOTO 63260
63170  GOSUB 63300
63180  PRINT
63190  FOR Z6 = 0 TO  PEEK (Z9 + 6) + 256 *  PEEK (Z9 + 5) - 1
63200  PRINT "ELEMENT (";Z6;") = ";
63210  IF (Z4) THEN  PRINT Z3%(Z6): GOTO 63240
63220  IF (Z5) THEN  PRINT Z3$(Z6): GOTO 63240
63230  PRINT Z3(Z6)
63240  NEXT Z6
63250  GOSUB 63310
63260  GOSUB 63390: GOTO 63140
63270 Z1 =  PEEK (Z9):Z2 =  PEEK (Z9 + 1)
63280 Z4 = Z1 > 127:Z5 = Z2 > 127
63290  RETURN
63300  POKE Z9 + 1,51 + 128 * Z5: POKE Z9,90 + 128 * Z4: RETURN
63310  POKE Z9,Z1: POKE Z9 + 1,Z2: RETURN
63320 Z0 = 0
63330  IF  NOT (Z1 = 90 OR Z1 = 138) THEN  RETURN
63340  IF ((Z2 - 128 * Z5) > 47) AND ((Z2 - 128 * Z5) < 58) THEN
    Z0 = 1
63350  RETURN
63360 Z9 =  PEEK (105) + 256 *  PEEK (106)
63370 Z8 =  PEEK (107) + 256 *  PEEK (108)
63380  RETURN
63390 Z9 = Z9 +  PEEK (Z9 + 2) + 256 *  PEEK (Z9 + 3): RETURN
63400  PRINT  CHR$ (Z1); CHR$ (Z2);
63410  IF (Z4) THEN  PRINT "%";: RETURN
63420  IF (Z5) THEN  PRINT "$";
63430  RETURN
63440 Z8 =  PEEK (109) + 256 *  PEEK (110): RETURN
```

Flashing Cursor

If you have an 80-column Apple IIe, IIc or IIGS, you can use this short program to customize your cursor. Completely invisible to your application program, this routine works in both 40- and 80-column modes.

by Cecil Fretwell

The "Apple Presents . . . Apple" disk that comes with the Apple IIe demonstrates a flashing cursor. The *80-Column Text Card Manual* doesn't show a control code to turn on the flashing cursor. So how do you set a flashing cursor? The answer is that no such control code exists. A special program must be written to obtain a flashing cursor. Personally, I prefer a flashing cursor, and writing such a program was a challenge I abandoned many times before I finally succeeded.

Once installed, the Flashing Cursor program works well no matter which screen mode is active, or whether you're using BIG MAC, PLE, or almost any other program or utility. The flashing underline character replaces the active cursor, whether it is the flashing checkerboard or the solid steady inverse cursor if the card is active. For example, suppose the cursor is positioned at an A on the screen. What you will see using Flashing Cursor is an alternate display of the A and the underline character.

LIMITATIONS

Unfortunately, the code does not produce a flashing cursor 100% of the time. First consider the flashing checkerboard mode. I call this the regular 40-column mode, since it is exactly like the Apple II Plus mode. The only flaw in this mode occurs when you perform the CATALOG command, and the display stops to allow you to examine the file names on the screen. At this point, the flashing checkerboard cursor appears.

When you complete the CATALOG command, the flashing underline cursor returns. To replace the flashing checkerboard cursor in this instance is not impossible, but it would require twice the code.

When the card is active, the flaw when you do a CATALOG still exists. The difference is that a solid cursor appears during the CATALOG command pauses. A second flaw occurs when escape mode is invoked. Pressing <ESC> causes the cursor to be replaced with a nonflashing plus (+) character. When you exit from escape mode, the solid cursor is displayed until you move it from the position at which the escape mode was exited. Nothing can be done to flash the plus character because escape mode is handled entirely within the ROM code on the card. Other than these exceptions, the system works exactly as if the Flashing Cursor code were not installed.

ENTERING THE PROGRAM

Applesoft Version

To key in the program, type in the FLASHING.CURSOR program as shown in **Listing 36** and save it to disk with the command:

```
SAVE FLASHING.CURSOR
```

Before you RUN the program for the first time, I strongly suggest that you remove the disks from your drive(s), or at least open the doors of the drives. The process of installing FLASHING.CURSOR involves fooling "Mother DOS," who gets extremely unhappy if you make a mistake.

If you get the "ADDRESS FOR FLASHING CURSOR?" prompt, there is a good chance that you entered the DATA statements correctly. If the program results in the message DATA

STATEMENTS ARE WRONG, you have made a mistake entering the DATA statements in **lines 180-330**. Correct your mistake(s), SAVE and RUN the program again.

Now where do you want the FLASHING.CURSOR code located? Suppose you choose the well-worn area at $300. Enter 768 (the decimal equivalent of $300) and press <RETURN>. You should get the message FLASHING CURSOR IS INSTALLED, along with the display of a flashing cursor. If the system dies, you may have made a mistake in **lines 130-150**, or the address you specified for the flashing code wiped out DOS or other valuable code. (See why I urged you to save the program and protect your disks before you test it?)

The location for the code is up to you. If you have other programs you want to load into $300, you will have to find 151 bytes of free space somewhere else. Just determine the decimal equivalent of the beginning address and enter this value at the "ADDRESS FOR FLASHING CURSOR?" prompt. After determining that the Applesoft program is correct, you can now safely RUN it from disk, LOAD and RUN it, include it in your HELLO program, etc. Now test your program. I hope you will find no flaws other than those mentioned earlier.

Machine Language Version

Please refer to Appendix A for help in entering CURSOR.ML (**Listing 37**). If you key it in from the Monitor, save it to disk with the command:

```
BSAVE CURSOR.ML, A$2EE, L$A9
```

If you want to store the machine language program in an area of memory other than that starting at $300, enter the hex code (starting at **line 57**) into the desired area of memory. Then enter the code in **lines 39-45** starting at $2EE, replacing location $2F4 with the low address byte of the code, and location $2F9 with the high address byte of the code. Before you test your work, save it to disk as two separate files with names of your own choice.

The first, consisting of the FLASHING CURSOR code from **line 57** on, should be saved using the address parameter of the location at which the code was entered, and a length parameter of $97. The second file will contain the code at $2EE and will have an address parameter of $2EE and a length parameter of $12. Then BLOAD the FLASHING CURSOR code, and BRUN the $2EE code.

Do not attempt to short circuit the $2EE code! For example, do not try to BLOAD the FLASHING CURSOR code, and then use the ROM monitor to plant the code at $9EBA. If you do, "Mother DOS" will die with unpredictable results!

HOW IT WORKS

The key to success is the hook planted at $9EBA. DOS traps all characters by replacing KSWL, KSWH with the $9E81 address. After some gyrations, KSWL and KSWH are replaced with the proper hook to the keyboard code, and DOS performs a JMP (KSWL) to that code.

Once a key is pressed, KSWL and KSWH are set back to $9E81. Trying to replace the true KSWL (e.g., $FD1B for regular 40-column mode) retained by DOS at $AA55, by a hook to the FLASHING CURSOR code, which then does the JMP (KSWL) ($9EBA in DOS) won't do the trick. For example, a PR#3 from regular 40-column will disable the FLASHING CURSOR code.

In order to make the code position independent and to minimize the amount of space it requires, existing subroutines in the 80-column card such as SCREEN were used. This requires careful switching in and out of the card firmware (**lines 57-61** and **lines 128-131**). This concept was borrowed from the F8 ROM listing. The rest of the code requires an understanding of how characters are retrieved and placed on the screen.

Lines 57-77 perform two functions. First, they save the state of the system before Flashing Cursor is activated and wait for a key to be pressed. After a key is pressed, the system state is restored, and the applications software is not even aware of the manipulations being performed.

Next, the ISTAT location must be set up for use by KEYDLY. **Lines 58-94** operate with interrupts disabled. Once KEYDLY comes into play, interrupts are allowed to occur while Flashing Cursor is active.

Lines 78, 79 and 80 may seem unnecessary. This code appears in a lot of Apple's software involving the IIe 80-column card. The F8 ROM code that was executed before reaching Flashing Cursor tried to replace the character at the current cursor position with a flashing character. Not only does this foul up 80-column mode, but also, because of the way the IIe maintains the screen buffers, failure to perform **lines 78, 79 and 80** would result in an unwanted character on the screen.

Having reached **line 81**, this is a good time to explain the function of RD80VID. If bit 7 of this location is on, we know that the text card is active and in 80-column mode. If it is off, we are in 40-column mode and at this point do not care whether the card is active or not. If 80-column mode is active, **lines 83-86** repair the improper screen character fetched by the F8 ROM software.

The subroutine SCREEN is another important feature on the text card. OURCH is the CH (horizontal position) value maintained by the text card. This pesky location is one of the reasons why things like HTAB and POKE 36,*xx* do not always correctly position the cursor on the screen. By loading the Y-Register with the current horizontal cursor position and clearing the V-flag, SCREEN will return the current screen character at the cursor in the A-Register. We need this character because it is the one we want to flash on the screen.

By loading Y with the current horizontal cursor position, loading A with the desired character, and setting the V flag, SCREEN will place the desired character on the screen. We will use this idea very shortly.

Finally, the cursor is flashed by having the current screen character alternate with the underline character. If you don't like the underline character, you can replace the contents of location 331 with whatever character you like. **Lines 87-117** perform the flashing work.

Lines 87-95 place the underline character on the screen then look for a key pressed via KEYDLY. RD80VID tells us how to set up the Y-Register properly so that SCREEN will place the underline character on the screen. KEYDLY not only lets interrupts occur, but also provides a delay, allowing the character in the A-Register to remain on the screen for a short time. If a key is pressed, Carry will be returned set and the character value of the key pressed will be in the A-Register. If a key is not pressed, Carry is returned clear.

For the moment, assume that no key is pressed; therefore, the branch in **line 96** is not taken. If you want to replace the underline character with the current character on the screen, this is where ALTCHR comes into play. If bit 7 of that location is turned on, the text card is active. If it is turned off, the 40-column mode is active.

If 40-column mode is active, no further work regarding the current character is required and the program branches to **line 109**. If the card is active, **lines 100-108** ensure that a flashing character is "replaced" by its nonflashing equivalent before control passes to **line 109**. Failure to perform this logic can produce some weird results on the screen.

When **line 109** is reached, the logic up to **line 116** performs the same operations as those performed for the underline character. If calling KEYDLY in **line 116** shows no key pressed, the program loops back to **line 89** to display the underline character.

When a key is finally pressed, either **line 96** or the failure to take the branch in **line 117** causes **lines 118-132** to be executed. **Lines 118-124** restore the original character to the screen. **Lines 125-131** restore the state of the system before Flashing Cursor was invoked. Finally, **line 132** proceeds through KSWL to get the key pressed, etc. At this point, the system doesn't even know that Flashing Cursor exists.

MODIFICATIONS

Since it can be customized, Flashing Cursor can be used in many creative ways. For instance, a program with several modes could use a different cursor for each mode. Use your imagination and have fun with Flashing Cursor!

LISTING 36: FLASHING.CURSOR

```
1  REM   **********************
2  REM   *  FLASHING.CURSOR    *
3  REM   * BY CECIL FRETWELL  *
4  REM   * COPYRIGHT (C) 1984 *
5  REM   * BY MICROSPARC, INC *
6  REM   * CONCORD, MA  01742 *
7  REM   **********************
10 S = 0
20  FOR I = 1 TO 152
30  READ C:S = S + C
40  NEXT I
50  IF S = 0 THEN 80
60  PRINT "DATA STATEMENTS ARE WRONG"
70  END
80  RESTORE
90  TEXT : HOME : VTAB 22: PRINT "** COPYRIGHT 1984 BY
     MICROSPARC, INC. **": VTAB 10: INPUT "ADDRESS FOR FLASHING
     CURSOR? ";AD%
100  FOR I = 0 TO 150
110  READ C: POKE I + AD%,C
120  NEXT I
130 H% = AD% / 256
140 L% = AD% - H% * 256
150  POKE 39625,L%: POKE 39835,L%: POKE 39626,H%: POKE 39836,H%
160  PRINT "FLASHING CURSOR IS INSTALLED"
170  END
180  DATA  8,120,44,21,192,8,141,7,192,133,252,104
190  DATA  168,104,72,42,42,42,42,133,253,152
200  DATA  72,165,252,72,138,72,165,252,164
210  DATA  36,145,40,173,31,192,16,9,172,123,5
220  DATA  184,32,6,207,133,252,169,223,164
230  DATA  36,44,31,192,16,3,172,123,5,44,88
240  DATA  255,32,6,207,36,253,32,198,194
250  DATA  176,47,165,252,44,30,192,48,17,164,50
260  DATA  192,127,208,11,201,64,144,7,201
270  DATA  128,176,3,24,105,64,164,36,44,31,192
280  DATA  16,3,172,123,5,44,88,255,32,6,207
290  DATA  36,253,32,198,194,144,184,165,252
300  DATA  164,36,44,31,192,16,3,172,123,5
310  DATA  44,88,255,32,6,207,104,170,104,40
320  DATA  48,3,141,6,192,40,108,56,0
330  DATA  -16049
```

LISTING 37: CURSOR.ML

SOURCE FILE -

```
                    ;
                    ;*********************************
                    ;                    CURSOR.ML
                    ;                     CUSTOM
                    ;                 FLASHING CURSOR
                    ;                  FOR APPLE IIE
                    ;                BY CECIL FRETWELL
                    ;                COPYRIGHT (C) 1984
                    ;                MICROSPARC, INC.
                    ;                CONCORD, MA  01742
                    ;
                    ;*********************************
                    ;
                    ;
                    CH       EQU $24          ;CURSOR COLUMN
                    BASL     EQU $28          ;CURSOR ADDRESS
                    INVFLG   EQU $32          ;INVERSE/FLASH/NORMAL
                    KSWL     EQU $38          ;KEYIN HOOK - LOW
                    OLDCHR   EQU $FC          ;ORIGINAL SCREEN CHAR
                    ISTAT    EQU $FD          ;INTERRUPT STATE
                    OURCH    EQU $57B         ;80 COLUMN CH
                    RDCHAR   EQU $9EBA        ;DOS JMP (KSWL)
                    SCXROM   EQU $C006        ;BANK STATUS
                    SETROMS  EQU $C007        ;SET ROMS ON
                    RDCXROM  EQU $C015        ;CURRENT ROM STATE
                    ALTCHR   EQU $C01E        ;READ ALT CHAR SWITCH
                    RD80VID  EQU $C01F        ;READ 80 COLUMN SWITCH
                    KEYDLY   EQU $C2C6        ;KEY DELAY SUBROUTINE
                    SCREEN   EQU $CF06        ;PICK/STORE SCREEN
                    SEV      EQU $FF58        ;KNOWN RTS
                    ;
                    ;
                    ;
                    ; SET UP TO FOOL MOTHER DOS
                    ;
                    ;
                    ;
                             ORG $2EE
                    ;
02EE  A9 4C                  LDA #$4C         ;PLANT JMP TO US
02F0  8D BA 9E               STA RDCHAR       ;FROM DOS
02F3  A9 01                  LDA #<FLASH
02F5  8D BB 9E               STA RDCHAR+1
02F8  A9 00                  LDA #>FLASH
02FA  8D BC 9E               STA RDCHAR+2
02FD  4C D0 03               JMP $3D0         ;RETURN TO DOS
                    ;
                    ;
                    ; ALL OF THE WORK IS DONE HERE
                    ; NOTE THAT CODE IS POSITION
                    ; INDEPENDENT, THEREFORE, IT
                    ; CAN BE MOVED TO ANY DESIRED
                    ; AREA OF MEMORY WITH A
                    ; CORRESPONDING CHANGE TO
```

```
                        ; LOCATIONS $9EBA-$9EBC
                        ;
                        ;
0300   08               FLASH    PHP              ;SAVE USER IRQ STATE
0301   78                        SEI              ;INHIBIT DURING BANK SWITCH
0302   2C 15 C0                  BIT RDCXROM      ;GET CURRENT STATE
0305   08                        PHP              ;SAVE ROM BANK STATE
0306   8D 07 C0                  STA SETROMS      ;SET ROMS ON
0309   85 FC                     STA OLDCHR       ;SAVE A
030B   68                        PLA              ;HOLD ONTO CXBANK STATUS
030C   A8                        TAY
030D   68                        PLA              ;A EQU USER'S IRQ STATE
030E   48                        PHA              ;AND RETAIN ON STACK
030F   2A                        ROL              ;MOVE IRQ BIT TO V BIT
0310   2A                        ROL
0311   2A                        ROL
0312   2A                        ROL
0313   85 FD                     STA ISTAT        ;SAVE FOR KEYDLY
0315   98                        TYA              ;PUT CXBANK STATUS
0316   48                        PHA              ;  BACK ONTO STACK
0317   A5 FC                     LDA OLDCHR       ;SAVE OLD A ON STACK
0319   48                        PHA
031A   8A                        TXA              ;SAVE X REGISTER
031B   48                        PHA
031C   A5 FC                     LDA OLDCHR       ;REPAIR MONITOR'S
031E   A4 24                     LDY CH           ;SILLY ATTEMPT
0320   91 28                     STA (BASL),Y
0322   AD 1F C0                  LDA RD80VID      ;80 COLUMN ACTIVE?
0325   10 09                     BPL KEYLOOP      ;IF NOT
0327   AC 7B 05                  LDY OURCH        ;THROW AWAY A FROM
032A   B8                        CLV              ;DOS AND REPLACE
032B   20 06 CF                  JSR SCREEN       ;WITH SCREEN CHAR
032E   85 FC                     STA OLDCHR
0330   A9 DF            KEYLOOP  LDA #$DF         ;UNDERLINE CHARACTER
0332   A4 24                     LDY CH           ;ASSUME 40 COL
0334   2C 1F C0                  BIT RD80VID
0337   10 03                     BPL KEY1         ;IF 40 COLUMN MODE
0339   AC 7B 05                  LDY OURCH        ;USE 80 COL CH
033C   2C 58 FF         KEY1     BIT SEV          ;STORE ON SCREEN
033F   20 06 CF                  JSR SCREEN       ;DISPLAY UNDERLINE CHAR
0342   24 FD                     BIT ISTAT        ;SET UP INTERRUPT STATE
0344   20 C6 C2                  JSR KEYDLY       ;LOOK FOR KEY
0347   B0 2F                     BCS FLASHR       ;IF GOT KEY
0349   A5 FC                     LDA OLDCHR       ;OLD CHARACTER
034B   2C 1E C0                  BIT ALTCHR       ;OLD 40 COLUMN?
034E   30 11                     BMI KEYF         ;IF NOT
0350   A4 32                     LDY INVFLG
0352   C0 7F                     CPY #$7F         ;FLASHING?
0354   D0 0B                     BNE KEYF         ;IF NOT
0356   C9 40                     CMP #$40         ;FLASHING CHAR?
0358   90 07                     BCC KEYF         ;IF NOT
035A   C9 80                     CMP #$80         ;FLASHING CHAR?
035C   B0 03                     BCS KEYF         ;IF NOT
035E   18                        CLC              ;MAKE IT NORMAL
035F   69 40                     ADC #$40         ;CHARACTER
0361   A4 24            KEYF     LDY CH           ;ASSUME 40 COL
0363   2C 1F C0                  BIT RD80VID
0366   10 03                     BPL KEY2         ;IF 40 COL
```

```
0368  AC 7B 05              LDY OURCH       ;USE 80 COL CH
036B  2C 58 FF   KEY2       BIT SEV         ;STORE OLD CHARACTER
036E  20 06 CF              JSR SCREEN      ;ON SCREEN
0371  24 FD                 BIT ISTAT       ;SET UP INTERRUPT STATE
0373  20 C6 C2              JSR KEYDLY      ;LOOK FOR KEY
0376  90 B8                 BCC KEYLOOP     ;IF NO KEY YET
0378  A5 FC      FLASHR     LDA OLDCHR      ;RESTORE OLD CHARACTER
037A  A4 24                 LDY CH          ;ASSUME 40 COL
037C  2C 1F C0              BIT RD80VID
037F  10 03                 BPL KEY3        ;IF 40 COL
0381  AC 7B 05              LDY OURCH       ;USE 80 COL CH
0384  2C 58 FF   KEY3       BIT SEV         ;STORE OLD CHARACTER
0387  20 06 CF              JSR SCREEN      ;ON SCREEN
038A  68                    PLA             ;RESTORE X
038B  AA                    TAX
038C  68                    PLA             ;RESTORE A
038D  28                    PLP             ;GET PRIOR I/O STATE
038E  30 03                 BMI FLASH1      ;IF NO BANK RESTORE
0390  8D 06 C0              STA SCXROM      ;RESTORE BANK
0393  28         FLASH1     PLP             ;RESTORE IRQ
0394  6C 38 00              JMP (KSWL)      ;CONTINUE THRU DOS

000  ERRORS

02EE  HEX START OF OBJECT
0396  HEX END OF OBJECT
00A9  HEX LENGTH OF OBJECT
9539  HEX END OF SYMBOLS
```

Auto Date

You may not have a clock card in your Apple, but supplying the date when your boot with this short Hello program will give your other programs access to the date for as long as your Apple has power.

by Clay Carr

Have you ever wished that your Apple had a built-in date function? Of course, you can buy a clock card to get it. Or you can use this short Applesoft routine that will do almost as much.

The routine in DATE.HELLO (Listing 1) can be included in your Hello program. What is does it quite simple: it takes today's month, day and year (or any month, day and year) and POKEs them into the last three available locations of memory page 3. Normally, these locations aren't used by running programs, so the date is available no matter how many programs you've run since you stored it there.

To key in the program, type in the **DATE.HELLO** program as shown in **Listing 38,** and save it to disk with the command:

SAVE DATE.HELLO

USING THE PROGRAM

After running DATE.HELLO, there are any number of ways that you can use the date. The simplest and most useful is to put this instruction toward the beginning of a program:

```
100 DT$ = PEEK (973) + "/" + PEEK (974) + "/" + PEEK (975)
```

Note: do not use DATE$ as the variable. If you do, you'll find that the Apple has parsed it into D AT E$ — which will send your program crashing in flames.

The DATE.HELLO program doesn't store the date as a string (though it would be easy to modify it to do this) because you may want to use the month, day and year values in computations. For instance, if you want to use the banker's 360-day year to compute elapsed time, you might want to use the data this way:

```
100 DT = PEEK (973) * 30 + PEEK (974) + PEEK (975) * 360
```

STORING THE DATE

Locations 973-975 ($3CD-$3CF) are not the only places that you could safely store the date — they are just the quickest and easiest ones. Other bits and pieces of unused space are scattered throughout the Apple, from page 0 to DOS 3.3 (such as $9CF8-$9CFF). If you're a machine language programmer, you'll probably want to place the date into one of these and leave all of page 3 available.

Caution: There are may different utilities that use the vacant locations in page 0, page 3 and other normally available spots. If you use such utilities, you'll need to experiment to find the places in which data can safely be POKEd.

LISTING 38: DATE.HELLO

```
20  REM   **********************
30  REM   *     DATE.HELLO     *
40  REM   *    BY CLAY CARR    *
50  REM   * COPYRIGHT (C) 1984 *
60  REM   * BY MICROSPARC, INC *
70  REM   * CONCORD, MA  01742 *
80  REM   **********************
90  REM
100  TEXT : HOME : PRINT "** COPYRIGHT 1984 BY MICROSPARC, INC.
     **": VTAB 5: PRINT "DO YOU WANT TO STORE AN ALPHANUMERIC":
     PRINT "DATE FOR USE IN YOUR PROGRAMS TODAY?": VTAB 8: HTAB
     18: GET Y$: ON 1 + (Y$ = "N") + 2 * (Y$ = "Y") GOTO 100,190
110  TEXT : HOME : VTAB 5: HTAB 3: PRINT "INPUT THE DATE THAT
     YOU WANT TO USE:"
120 D$ = "": VTAB 8: HTAB 16: CALL  - 868: PRINT "  /  / "
130  VTAB 8: HTAB 16: GET D$: PRINT D$;:MO$ = D$: GET D$: PRINT
     D$"/";:MO =  VAL (MO$ + D$)
140  GET D$: PRINT D$;:DA$ = D$: GET D$: PRINT D$"/";:DA =  VAL
     (DA$ + D$)
150  GET D$: PRINT D$;:YR$ = D$: GET D$: PRINT D$;:YR =  VAL
     (YR$ + D$)
160  ON 1 + (MO > 1 AND MO < 13) * (DA > 0 AND DA < 32) GOTO 120
170  HTAB 25: PRINT "  OK? ";: GET Y$: ON 1 + (Y$ = "N") + 2 *
     (Y$ = "Y") GOTO 160,120
180  POKE 973,MO: POKE 974,DA: POKE 975,YR
190  END
```

Free Sector Chart

This short Applesoft program explores the way machine language concepts can be written in BASIC, while it provides an easy way to chart the amount of free space on a DOS 3.3 disk.

by Donald Jessop

Recently, a friend and I were discussing why all of the programs that calculate the free space on a disk seem to be written in machine language. My friend assumed that such a program would be almost impossible to write in Applesoft. In my attempt to prove him wrong, I wrote Free Sector Chart (**Listing 39**). In addition to calculating the amount of free space on a disk, Free Sector Chart presents a chart of the disk showing which sectors have data on them.

THE CONVERSION TECHNIQUE

Since DOS stores the vacancy information in two bytes in the Volume Table of Contents (VTOC), there were thousands of possible combinations of filled and unfilled sectors. I realized that I would have to examine the information bit by bit.

The technique I used was to go from high to low bit, checking to see if the value from the byte exceeded a specific value. For example, if the seventh bit was set, the value of the byte would be equal to or greater than 2^7 or 128. If we subtract 128 from this byte, we can ignore the seventh bit and concentrate on the sixth bit. The loop in **lines 330-370** does just this by stepping backwards through a FOR loop. By using this technique, many complicated machine language subroutines can be converted to BASIC, thus enabling you to use the subroutine on almost any machine.

ENHANCEMENTS

By adding a subroutine to dump the contents of the screen to a printer, Free Sector Chart can be used to keep a close eye on how much space you have available in your disk library. Adding a catalog listing option would make this an excellent archival aid.

LISTING 39: FREE.SECTOR.CHART

```
1  REM  ***********************
2  REM  *  FREE.SECTOR.CHART  *
3  REM  *  BY DONALD JESSOP   *
4  REM  * COPYRIGHT (C) 1984  *
5  REM  * BY MICROSPARC INC   *
6  REM  * CONCORD, MA  01742  *
7  REM  ***********************
10  REM ** LOAD IN RWTS SUBROUTINE **
20  FOR X = 896 TO 896 + 30: READ D: POKE X,D: NEXT
30  DATA
     169,3,160,138,32,217,3,96,0,0,1,96,1,0,17,0,153,3,0,32,0,0,
     1,0,0,96,1,0,1,239,216
40 A$ = "0123456789ABCDEF"
50  HOME : PRINT : PRINT "** COPYRIGHT 1984 BY MICROSPARC, INC.
    **"
60  INVERSE : HTAB 10: PRINT "FREE SECTOR CHART": NORMAL : PRINT
70  INPUT "WHAT IS THE NAME OF THE DISK?              ";NA$
80 NA$ = "FREE SECTOR CHART FOR " + NA$
```

```
90 A =  LEN (NA$): IF A < 39 THEN  GOTO 110: REM ** WE ARE
    CHECKING TO SEE IF THE TITLE CAN BE CENTERED PROPERLY **
100 NA$ =  LEFT$ (NA$,38)
110  PRINT "WOULD YOU ALSO LIKE A CATALOG OF THE    DISK? ";:
     GET P$: PRINT P$
120  IF P$ = "Y" THEN  PRINT  CHR$ (4)"CATALOG"
130  PRINT : INVERSE : PRINT "PRESS ANY KEY TO CONTINUE";:
     NORMAL : GET B$: PRINT B$
140  CALL 896: REM ** WE NOW READ IN THE VTOC **
150  REM ** DRAW BORDER FOR CHART **
160  HOME
170  VTAB 1: HTAB (38 -  LEN (NA$)) / 2 + 1: PRINT NA$
180  INVERSE
190  VTAB 3: HTAB 2: PRINT A$;A$; LEFT$ (A$,3)
200  FOR X = 1 TO 16: VTAB 3 + X: HTAB 1: PRINT  MID$ (A$,X,1);:
     NORMAL : PRINT  SPC( 35);: INVERSE : PRINT  MID$ (A$,X,1):
     NEXT
210  HTAB 2: PRINT A$;A$; LEFT$ (A$,3)
220  NORMAL
230  REM ** DETERMINE WHICH SECTORS ARE FILLED **
240 H = 1:P = 0
250  FOR X = 8247 TO 8383 STEP 4
260 H = H + 1
270 V = 20
280 P = P + 1: IF P > 16 THEN P = 1
290  VTAB V: HTAB H: FLASH : PRINT  MID$ (A$,P,1);: NORMAL
300  FOR Y = 1 TO 2
310 A =  PEEK (X + Y)
320  REM ** THIS LOOP EXTRACTS THE INFORMATION BIT BY BIT **
330  FOR T = 7 TO 0 STEP  - 1: REM ** WE STEP BACKWARD THROUGH
     THE BYTE **
340 V = V - 1: VTAB V: HTAB H
350  IF A < (2 ^ T) THEN  PRINT "*"
360  IF A > (2 ^ T) - 1 THEN A = A - 2 ^ T:F = F + 1
370  NEXT
380  NEXT
390  VTAB 20: HTAB H: INVERSE : PRINT  MID$ (A$,P,1);
400  NEXT
410  NORMAL
420  VTAB 22: HTAB 1: PRINT "THERE ARE ";F;" FREE SECTORS"
430  END
```

ProDOS RESET Trap

Trapping the <RESET> key under ProDOS can be smoothly handled by using the ONERR routine. A sample program shows you how it's done.

by Eric Seiden

While developing an interactive database program, I found it necessary to ensure that the only exit route was through the save data routine, rather than by pressing <CTRL>C or <RESET>. Since <CTRL>C causes an Applesoft error that can be trapped with the ONERR statement, all that remained was to disable <RESET>. Under DOS 3.3, this simply required two POKEs (POKE 40286,252: POKE 40287,164), but under ProDOS, it is a little more complicated.

TRAPPING RESET UNDER PRODOS

The method presented here uses POKEs to point the Reset vector at a short machine language routine that can reside anywhere in memory. This routine then loads an error code value into the Accumulator and jumps to the ProDOS error handler. That way, your program can intercept RESETs as errors by using the ONERR statement and identify them by their code number. This value may be obtained with the statement PEEK(222).

The error routine for ProDOS starts at $BE09. It is very simple to give the Reset vector any error code that you want. The following is a machine language routine to accomplish this task:

```
350: A9 3E      LDA #$3E
352: 20 09 BE   JMP $BE09
```

For demonstration purposes, the routine is located at $350, but it may be relocated anywhere that it will not be overwritten. To point the Reset vector to this routine, use the following statement:

```
POKE 1010,LB:POKE 1011,HB:CALL-1169
```

where *LB* is the decimal value of the low byte of the address for the machine language routine, and *HB* is the decimal value of the high byte. (The CALL -1169 simply sets a "power-up" byte to let the Apple know that the Reset vector is legitimate.)

To use this technique in a program, it is necessary to POKE the machine language routine in memory, change the Reset vectors and include an ONERR-GOTO statement and an error-handling routine. A demonstration of this technique is shown in **Listing 40**.

THE DEMONSTRATION PROGRAM

When you run the program, it will give you the choice of installing the RESET trap, removing the RESET trap or quitting. If you choose option 1 to install the trap, the machine language routine will be POKEd into memory and the RESET vectors changed. The program will then proceed to count to one thousand until interrupted by a RESET. At this point, it will print the error number and wait for a keypress before returning to the menu.

Option 2 will remove the trap before counting to one thousand. This may be confirmed by RESETting out of the program. Option 3 will restore your system to normal before quitting to be certain that the trap is removed.

ENTERING THE PROGRAM

To key in the program, enter it as shown in **Listing 40** and save it to disk with the command:

SAVE TRAP.RESET

CUSTOMIZING THE TRAP

Any value may be assigned to RESET errors by changing the value at $351 in the machine language routine (or the POKE 849 in **line 200**). For instance, if you know that you want RESET to be interpreted as a particular ProDOS error, you could simply assign that error value to it. If no ONERR statement is active when <RESET> is pressed, the word ERROR will be printed. This may be confirmed by deleting **line 220**, choosing option 1, and pressing <RESET>.

Even under control of the trap, a RESET will cause the cursor to be relocated at the bottom of the screen. This should be taken into consideration when designing error message displays, or when rerouting program flow back into the main body of the program.

LISTING 40: TRAP.RESET

```
10  REM  ************************
20  REM  *       TRAP.RESET     *
30  REM  *    BY ERIC SEIDEN    *
40  REM  *  COPYRIGHT (C) 1984  *
50  REM  *  BY MICROSPARC, INC  *
60  REM  *  CONCORD, MA. 01742  *
70  REM  ************************
80  HOME : PRINT "CHOOSE:": PRINT " <1> RESET TRAPPED": PRINT "
     <2> RESET NORMAL": PRINT " <3> QUIT"
90  HTAB 5: GET K$: PRINT K$:K =  VAL (K$): IF K < 1 OR K > 3
     THEN 90
100  IF K = 3 THEN  GOSUB 250: HOME : END
110  IF K = 1 THEN  GOSUB 200: GOTO 130
120  IF K = 2 THEN  GOSUB 250: GOTO 130
130  HOME : VTAB 10: PRINT "RESET IS ";
140  IF K = 1 THEN  PRINT "TRAPPED."
150  IF K = 2 THEN  PRINT "NORMAL."
160  FOR I = 1 TO 1000: VTAB 15: CALL  - 958: PRINT I: NEXT
170  GOTO 80
180  PRINT "ERROR NO.:"; PEEK (222): PRINT "PRESS A KEY TO
     CONTINUE": GET K$: PRINT : GOTO 80
190  REM  INSTALL RESET TRAP
200  POKE 848,169: POKE 849,62: POKE 850,32: POKE 851,9: POKE
     852,190: REM  POKE ML ROUTINE AT $350
210  POKE 1010,80: POKE 1011,3: CALL  - 1169: REM   SET RESET
     VECTOR TO POINT AT ML ROUTINE
220  ONERR  GOTO 180
230  RETURN
240  REM  REMOVE RESET TRAP
250  POKE 1010,0: POKE 1011,190: CALL  - 1169
260  RETURN
```

Shades and Textures

This short program displays different color combinations available on the Apple Hi-Res screen.

by Ted Huntington

To help explore the range of colors and textures available on the Apple, I wrote a short demonstration program called SHADES.TEXTURES. It displays the full range of colors available on the Hi-Res screen in pairs, with one color next to another. It not only displays the different color combinations possible, but also the textures that result from their interaction. The program could aid in designing the screens for your next Hi-Res graphics project.

USING THE PROGRAM

When you run SHADES.TEXTURES, you are given the choice of automatically cycling through pairs of colors, manually selecting the two colors to view, or quitting. If you choose to manually select colors, you will next be asked to specify two numbers from 0-7. After these are displayed, you are prompted to enter two new numbers. The automatic mode will cycle through the colors, pausing after each combination is displayed until a key is pressed. Press <ESC> to return to the menu if you want to switch modes.

ENTERING THE PROGRAM

To key in the program, enter **Listing 41** as shown and save it to disk with the command:

SAVE SHADES.TEXTURES

LISTING 41: SHADES.TEXTURES

```
10   REM  ************************
20   REM  *     SHADES.TEXTURES   *
30   REM  *     TED HUNTINGTON    *
40   REM  *   COPYRIGHT (C) 1984  *
50   REM  *   BY MICROSPARC, INC  *
60   REM  *   CONCORD, MA. 01742  *
70   REM  ************************
80   ONERR  GOTO 120
90   HOME : VTAB 5: HTAB 12: INVERSE : PRINT " SHADES AND
      TEXTURES ": NORMAL : PRINT : HTAB 13: PRINT "BY TED
      HUNTINGTON": PRINT : PRINT : HTAB 8: PRINT "<PRESS ANY KEY
      TO BEGIN >": VTAB 22: PRINT "** COPYRIGHT 1984 BY
      MICROSPARC, INC. **"
100   VTAB 15: HTAB 20: GET AA$
110   HGR
120   HOME : VTAB 21: PRINT "AUTO OR MANUAL OR QUIT (A/M/Q):";:
      GET AA$: IF AA$ < > "A" AND AA$ < > "M" AND AA$ < > "Q"
      THEN 120
130   PRINT : IF AA$ = "A" THEN 240
140   IF AA$ = "Q" THEN  TEXT : HOME : END
150  NY = 0
160   GOTO 190
170   FOR X = 1 TO 140 STEP 3: HCOLOR= C: HPLOT 0,X TO X,0:
      HCOLOR= D: HPLOT 3,X TO X,0: NEXT
```

```
180  IF NY = 1 THEN  RETURN
190  HOME : VTAB 21: CALL  - 958: PRINT "FIRST COLOR:";: GET
     C$:C =  VAL (C$): PRINT C: IF C < 0 OR C > 7 THEN 190
200  IF C$ =  CHR$ (27) THEN 120
210  VTAB 22: CALL  - 958: PRINT "SECOND COLOR:";: GET D$:D =
     VAL (D$): PRINT D: IF D < 0 OR D > 7 THEN 210
220  IF D$ =  CHR$ (27) THEN 120
230  GOTO 170
240 NY = 1: FOR E = 0 TO 7: FOR F = 0 TO 7
250  HOME : VTAB 22: PRINT "COLORS-";C;" AND ";D;""
260  GOSUB 170
270 C = C + 1
280  GET AA$
290  IF AA$ =  CHR$ (27) THEN 120
300  NEXT F
310 C = 0:D = D + 1
320  NEXT E
330  GOTO 20
```

Auto Case Convert

Have you converted your Apple to display lowercase text, but find yourself with lots of data files that were created in the days of all uppercase? This short machine language routine will automatically account for initial capitals and convert the rest of your text to lowercase.

by Bruce E. Howell, D.D.S.

When I finally installed a keyboard enhancer and a lowercase chip in my Apple II Plus, I found that I was stuck with many name and address files in stuffy, mechanical-looking uppercase. So I created a program to convert the uppercase. Instead of trying to modify a wide variety of data file formats, I allow the existing programs to print the data and modify the appearance of the output. Although a lowercase chip is required to display upper and lowercase on the screen, no hardware modification is required to output upper and lowercase data to a printer.

USING THE PROGRAM

To use CASE.CONVERTER (**Listing 42**), either BRUN the program or BLOAD it and issue a CALL 768 from Applesoft. Neither <CTRL><RESET> nor PR#0 turns off the program, but a CALL 787 will reset all required hooks and pointers.

ENTERING THE PROGRAM

Please refer to Appendix A for help in entering this program. If you key it in from the Monitor, save it to disk with the command:

BSAVE CASE.CONVERTER,A$300,L$71

HOW IT WORKS

The program follows a simple rule: convert all alphabetic characters to lowercase unless the character is preceded by a non-alpha character. The characters are intercepted on their way to the Monitor COUT1 routine by changing the hook at $36 and $37. (It also modifies a DOS pointer that changes $36 and $37.) The incoming character is checked; if it is not an alpha character, FLAG1 is reset to zero. If it is alphabetic and FLAG1 is not zero, the character is converted to lowercase by XORing with $20. An alpha character also sets FLAG1 to one for the next pass.

If converted, DOS commands from within an Applesoft program generate a syntax error. To avoid this, <CTRL>D sets FLAG2, which then passes all characters unchanged until a RETURN ($8D), thus resetting FLAG2 to zero.

LISTING 42: CASE.CONVERTER

```
SOURCE FILE -

                    ;
                    ; CASE.CONVERTER
                    ; BY BRUCE HOWELL
                    ; COPYRIGHT (C) 1984
                    ; MICROSPARC, INC.
                    ; CONCORD, MA 01742
                    ;
                              ORG $0300
 0300   A9 26       ON        LDA #START        ;SET POINTER TO START
```

```
0302  85 36               STA $36
0304  A5 03               LDA START/
0306  85 37               STA $37
0308  A9 26               LDA #START
030A  8D 04 9D            STA $9D04        ;SET DOS POINTER ALSO
030D  A5 03               LDA START/
030F  8D 05 9D            STA $9D05
0312  60                  RTS
0313  A9 BD      OFF      LDA #$BD         ;RESETS BOTH POINTERS
0315  85 36               STA $36          ;TO TURN IT OFF
0317  A9 9E               LDA #$9E
0319  85 37               STA $37
031B  A9 BD               LDA #$BD
031D  8D 04 9D            STA $9D04
0320  A9 9E               LDA #$9E
0322  8D 05 9D            STA $9D05
0325  60                  RTS
0326  8E 70 03   START    STX HOLD         ;SAVE X-REGISTER
0329  C9 8D               CMP #$8D         ;SEE IF RETURN
032B  F0 32               BEQ CR           ;YES, GO CHECK FLAG2
032D  AE 6F 03            LDX FLAG2        ;SEE IF DOS COMMAND
0330  E0 01               CPX #$01
0332  F0 1C               BEQ EXIT       ;DOS COMMAND - EXIT UNCHANGED
0334  C9 84               CMP #$84     ;CHECK FOR START OF DOS COMMAND
0336  F0 1F               BEQ DOS          ;YES IT IS
0338  C9 41               CMP #$41         ;SEE IF LESS THAN "A"
033A  10 0F               BPL NOALP        ;YES SO NON-ALPHA
033C  C9 5B               CMP #$5B         ;SEE IF GREATER THAN "Z"
033E  30 0B               BMI NOALP        ;YES SO NON-ALPHA
0340  AE 6E 03            LDX FLAG1        ;CHECK PREVIOUS CHAR.
0343  F0 02               BEQ DONT         ;DON'T CONVERT
0345  49 20               EOR #$20         ;CHANGE TO LOWER CASE
0347  A2 01      DONT     LDX #$01         ;SET FLAG1 TO ONE
0349  D0 02               BNE NEXT
034B  A2 00      NOALP    LDX #$00         ;RESET FLAG1 TO ZERO
034D  8E 6E 03   NEXT     STX FLAG1
0350  AE 70 03   EXIT     LDX HOLD         ;RESTORE X-REGISTER
0353  20 BD 9E            JSR $9EBD        ;GO PRINT IT
0356  60                  RTS
0357  A2 01      DOS      LDX #01          ;START OF A DOS COMMAND
0359  8E 6F 03            STX FLAG2
035C  4C 50 03            JMP EXIT
035F  AE 6F 03   CR       LDX FLAG2      ;COULD BE END OF DOS COMMAND
0362  E0 00               CPX #00
0364  F0 E1               BEQ DONT         ;NO, JUST PLAN CR
0366  A2 00               LDX #00
0368  8E 6F 03            STX FLAG2        ;YES, SO SET FLAG2
036B  4C 50 03            JMP EXIT
036E  00         FLAG1    DFC #00          ;PREVIOUS CHAR. FLAG
036F  00         FLAG2    DFC #00          ;DOS COMMAND FLAG
0370  00         HOLD     DFC 00           ;PROTECTS X-REGISTER

000  ERRORS

0300  HEX START OF OBJECT
0370  HEX END OF OBJECT
0071  HEX LENGTH OF OBJECT
95AD  HEX END OF SYMBOLS
```

Software Volume Control

The versatility of the Apple's single speaker is demonstrated by this short machine language routine that lets you control the volume of the tones generated, as well as the pitch and duration.

by Phil Goetz

While experimenting with a software voice for the Apple, I noticed that if I toggled the speaker twice in a row and then used a delay, the volume was much lower than if I had a delay between the two toggles. I found that this phenomenon can be used as a volume control. In most tone routines, you are allowed to specify duration and pitch. The program VOUMETONES (**Listing 43**) also allows you to specify volume.

USING THE PROGRAMS

To use VOLUMETONES, type BRUN VOLUMETONES. You must specify three parameters for each note: duration, pitch and volume. For duration, 0 is the longest, 255 ($FF) is the second longest, and 1 is the shortest. For pitch, 0 is the lowest, 255 is the second lowest, and 1 is the highest.

For volume, the maximum is half the number for the pitch, and the volume decreases as it approaches either one or one below the pitch. If the volume is set to zero, the same as the pitch, or greater than the pitch, a different note will be obtained, so be careful to keep the volume between one and one less than the pitch, inclusive.

Store the duration in memory location 769, the pitch in 771, and the volume in 788. Then, to play the note, CALL 768.

VOL.TONES.DEMO (**Listing 44**) is a demonstration program that plays a constant pitch starting at low volume, then reaches a maximum, and returns to a minimum. To sure to BLOAD VOLUMETONES before BRUNning VOL.TONES.DEMO.

ENTERING THE PROGRAMS

To key in the programs, type in **Listing 43** as shown and save it to disk with the command:

BSAVE VOLUMETONES,A$300,L$1F

Then enter **Listing 44** and save it to disk with the command:

BSAVE VOL.TONES.DEMO,A$1000,L$13

HOW VOLUMETONES WORKS

To understand how VOLUMETONES works, we must first know something about how the Apple creates sound. Locations $C030-$C03F are connected to the Apple speaker, which can be thought of as a cone. When any one of these locations is referenced, the speaker is pushed out; when addressed again, it returns to its original configuration. One of these times, a click is emitted. Therefore, every other time that a location from $C030-$C03F is addressed, the speaker makes a click.

When one of these locations, usually $C030, is addressed at a constant rate, a wave with a constant frequency (or pitch) is created. If $C030 is addressed at a faster rate, the frequency and thus the pitch, goes up.

Since the speaker only clicks every other time that it is addressed (or toggled), only every other toggle controls the pitch. If the second, silent toggle is midway between two audible clicks, the maximum volume will be reached. If it is closer to either audible toggle, the volume will be lower. The farther apart the toggles are, the greater the volume (to a point). This is why high-pitched tones on the Apple are lower in volume than low tones.

Here is my theory as to why this happens: The speaker, when addressed once, does not change its state instantly, but begins to move toward a reflexed position. If the next toggle follows closely after the first, it will pull back the speaker before it has reached the fully reflexed position. Therefore, the closer together the speaker toggles, the less the speaker movement and the smaller the vibration — thus the amplitude, and thus the volume.

LISTING 43: VOLUMETONES

```
                        ;
                        ;      VOLUMETONES
                        ;     BY PHIL GOETZ
                        ;   COPYRIGHT (C) 1984
                        ;   BY MICROSPARC, INC.
                        ;    CONCORD, MA 01742
                        ;
                                  ORG $300
0300  A0 90             DURATION LDY #$90        ;SET BY CALLER
0302  A9 60             PITCH    LDA #$60        ;SET BY CALLER
0304  38                         SEC
0305  ED 14 03                   SBC VOLUME+1
0308  8D 0C 03                   STA VT1+1
030B  A2 30             VT1      LDX #$30        ;PITCH MINUS VOLUME
030D  CA                VT2      DEX             ;DELAY
030E  D0 FD                      BNE VT2
0310  AD 30 C0                   LDA $C030       ;TOGGLE SPEAKER
0313  A2 30             VOLUME   LDX #$30
0315  CA                VT3      DEX             ;DELAY
0316  D0 FD                      BNE VT3
0318  AD 30 C0                   LDA $C030       ;TOGGLE SPEAKER
031B  88                         DEY      ;DECREMENT LOOP COUNTER (DURATION)
031C  D0 ED                      BNE VT1
031E  60                         RTS

000  ERRORS

0300  HEX START OF OBJECT
031E  HEX END OF OBJECT
001F  HEX LENGTH OF OBJECT
95CD  HEX END OF SYMBOLS
```

LISTING 44: VOL.TONES.DEMO

```
                        ;
                        ;    VOL.TONES.DEMO
                        ;     BY PHIL GOETZ
                        ;   COPYRIGHT (C) 1984
                        ;   BY MICROSPARC, INC.
                        ;   CONCORD, MA 01742
                        ;
                        ; NOTE: VOLUMETONES MUST BE LOADED
                        ;       BEFORE THIS DEMO IS RUN!
```

```
                       ;
                       ORG $1000
1000  A9 01            LDA #$01
1002  8D 14 03         STA $314       ; VOLUME IN DECIMAL 788
1005  20 00 03 VOLTONE JSR $300       ; CALL VOLUMETONES ROUTINE
1008  EE 14 03         INC $314       ; INCREASE VOLUME
100B  AD 14 03         LDA $314
100E  C9 60            CMP #$60       ; COMPARE TO MAXIMUM
1010  D0 F3            BNE VOLTONE    ; KEEP GOING
1012  60               RTS            ;DONE

000  ERRORS

1000  HEX START OF OBJECT
1012  HEX END OF OBJECT
0013  HEX LENGTH OF OBJECT
95F5  HEX END OF SYMBOLS
```

80-Column Catalog

Use this short DOS 3.3 patch to display a full 80-column CATALOG on your screen or print it out. Include the Applesoft routine in your Hello program or patch DOS for a permanent change.

by Robert C. Meltzer

While Pascal, FORTRAN, most assemblers, and even BASIC can take advantage of 80-column display capability, DOS 3.3 does not. This is most obvious when you execute a CATALOG command. The information scrolls up the left side of the screen while the right side goes to waste. I thought it would be useful to see twice the amount of catalog information on the screen. Since I was looking for an excuse to poke around inside DOS anyway, I took my copy of *Beneath Apple DOS* (an excellent reference by Quality Software) and started in. The result is 80-Column Catalog.

DISSECTING CATALOG

According to the detailed map of DOS (in Chapter 8 of *Beneath Apple DOS*), addresses \$AD98-\$AE2E (in a 48K machine) contain the CATALOG function handler. Disassemble 80 instructions starting from \$AD98, using any good disassembler or the L function of the Apple Monitor. With *Beneath Apple DOS* in hand you can follow the logic easily, assuming you understand the 6502 instruction set. Careful examination of the code, or of the *DOS manual*, indicates that one line of CATALOG output consists of:

1. The lock indicator (*).
2. The file type (A,I,B,T,S,R).
3. The file size in sectors.
4. The file name (up to 30 characters).

The item as printed is no more than 37 characters long. Two such items will fit on one line of an 80-column screen (or printer), as we shall see.

USING CATALOG80

Run the program CATALOG80 shown in **Listing 45**. This Applesoft program will modify DOS so that any future CATALOG commands will automatically result in an 80-column display. To get an 80-column printout, activate your printer before cataloging. Initializing a disk after running CATALOG80 will automatically transfer the 80-column catalog capability to the new disk.

DOS LINECHECKER

Listing 46 is the CATALOG function handler's LINECHECKER subroutine located at\$AE2F-\$AE41. This subroutine performs the following functions:

1. Sends a carriage return (CR) to the screen via the Monitor routine COUT at \$FDED.
2. Decrements a count once for each line sent to the screen.
3. Waits for a keypress when the count goes to zero (i.e., when the screen is full).
4. Reinitializes the line count after receiving the keypress.
5. Returns to the caller.

It also indicates other interesting locations in the CATALOG handler that affect the format of the screen output, specifically LINECHECKER calls that generate CRs between the DISK VOLUME header and the body of the catalog.

MODIFIED DOS LINECHECKER

The solution given in MODIFIED LINECHECKER (**Listing 47**) takes advantage of the fact that a screen display wraps at the end of each line. In other words, if after we send 80 characters to an 80-column screen, we send 80 more, there will be an implicit CR and the second 80 will be placed on the next line.

Each item put out by the CATALOG handler is exactly 37 characters. If, after each item, we send three blanks in place of the CR, this will result in exactly 80 characters of output per line, which will force an implicit CR after every two items.

Conveniently enough, there is a Monitor subroutine that sends three blanks to the output device — PRBLNK located at $F948. Simply by replacing the first two instructions at $AE2F of **Listing 46** with two NOPs followed by a JSR PRBLNK, we can send three blanks after each item instead of a CR, thus listing two items on each line. See **Listing 47**.

A couple of additional changes are required to complete this solution. The line count is currently initialized by LDA instructions located at $ADA3 and $AE3C. The value $16 allows 22 lines of output to the screen before waiting for a keypress. We'll use this count as an item count, rather than a line count, so we'll want its initial value set to $2A, to allow 42 items (two items on each of 21 lines) to be sent to the screen before waiting for a keypress.

We can't send a full 48 items (24 lines) to the screen because we want to leave the DISK VOLUME header at the top of the screen, and because each line we send forces a CR. Therefore, we must leave room for two lines at the top and a blank line at the bottom of the screen. Simply patching $2As into locations $ADA4 and AE3D$ will do the trick. Also,the JSRs at $ADC3 and ADC6 must call CROUT (the Monitor CR routine) instead of LNCHK, since LNCHK no longer sends CRs to the output device.

Listing 47 works, with no more increase in code space requirements than the original **Listing 46**. However, it will not function reliably with a printer since no CR or LF characters are ever sent by this routine to the output device.

THE BEST OF BOTH WORLDS

A solution that functions with an 80-column screen and equally well with a printer is given in **Listing 48**. This solution, being more general, requires more code. There are a few small unused areas of memory in DOS. We'll use the one at $BCDF to hold a patch to the CATALOG handler's LINECHECKER in **Listing 46**. Most of the code in the LINECHECKER remains unchanged. It doesn't routinely send a CR of course, but will still wait for a keypress if the screen is full. The major difference from the simple solution (**Listing 47**) is that it unconditionally sends two blanks (using the Monitor routine PRBL2 located at $F94A). Then it sends either a third blank or a CR, depending on whether it has just sent the left or the right item to the screen.

Determining each item's position (left or right) is the key. Th is accomplished by three instructions in the patch area. Since the item count is decremented once for each item output, its low-order bit changes once for each output. The LDA, ROR sequence gets the low-order bit of the item count in the C-bit of the P-Register, where it can be tested with the BCC instruction immediately following the ROR. If the bit is clear, the program sends a CR; if it's set, it just sends another space.

A couple of cosmetic changes complete our final solution. Since we're not implicitly sending a CR at the end of each line as in **Listing 47**, we can plug a $2C instead of a $2A into the item count and get more lines of text on the screen. We'll call CROUT instead of LNCHK as in **Listing 47**. Two instructions are added at $ADAA to tab the DISK VOLUME header to a more pleasing location on the screen.

If you have already used the $BCDF area of DOS for some other patch, there's another unused area of 45 bytes at $BA69. You need only change the address in the JMP instruction at $AE3F in **Listing 48**.

GETTING THE PATCH IN YOUR SYSTEM

There are several ways to get this patch into your system. You can include the program in **Listing 45** in your Hello program; this will simply put the patch in place each time you boot. Alternately, you can patch in the code using **Listing 45** or the Monitor, then INIT a new slave disk, that will then have the patch as a permanent part of its DOS. (This won't work for a master disk because the MASTER CREATE program reads a fresh copy of DOS into memory with which to create a master disk and ignores your nicely patched DOS.)

LISTING 45: CATALOG80

```
10  REM ***********************
20  REM *       CATALOG80     *
30  REM *  BY ROBERT MELTZER  *
40  REM *  COPYRIGHT (C) 1984 *
50  REM *  MICROSPARC, INC.   *
60  REM *  CONCORD, MA 01742  *
70  REM ***********************
80 HEX$ = "ADA4: 2C N ADA8: A9 13 85 24 EA EA N ADC3: 20 8E FD
     20 8E FD N D7D2G": GOSUB 120: CALL  - 144
90 HEX$ = "AE2F: EA CE 9D B3 D0 08 20 0C FD A9 2C 8D 9D B3 A2 02
     4C DF BC N D7D2G": GOSUB 120: CALL  - 144
100 HEX$ = "BCDF: 20 4A F9 A2 8D AD 9D B3 6A 90 02 A2 A0 8A 20
     ED FD 60 N D7D2G": GOSUB 120: CALL  - 144
110  END
120  FOR I = 1 TO  LEN (HEX$): POKE 511 + I, ASC ( MID$
     (HEX$,I,1)) + 128: NEXT I: POKE 72,0: RETURN
```

LISTING 46: LINECHECKER

```
                 ************************************
                 *           LINECHECKER            *
                 *   DOS CATALOG FUNCTION HANDLER   *
                 *             DOS 3.3              *
                 ************************************

                 COUT      EQU   $FDED       ; Monitor rtn to output (A)
                 RDKEY     EQU   $FD0C     ; Monitor keyboard wait routine

                           ORG   $ADA3
ADA3: A9 16                LDA   #$16        ; Initialize line count

                 * Code here prints out DISC VOLUME NNN

                           ORG   $ADC3
ADC3: 20 2F AE             JSR   LNCHK     ; Do a CR, dec line count, etc.
ADC6: 20 2F AE             JSR   LNCHK       ; Do the same again

               * Code here prints out info for each file & calls LNCHK

                           ORG   $AE2F
AE2F: A9 8D      LNCHK     LDA   #$8D        ; Load up a CR
AE31: 20 ED FD             JSR   COUT     ; Call Monitor routine to output
AE34: CE 9D B3             DEC   $B39D       ; Decrement line count
```

```
AE37: D0 08            BNE   $AE41      ; Branch if count not expired
AE39: 20 0C FD         JSR   RDKEY ; If expired, go wait for a keyin
AE3C: A9 15            LDA   #$15       ; Then reinitialize line count
AE3E: 8D 9D B3         STA   $B39D      ;   save for future reference
AE41: 60         RTRN  RTS              ;      return to caller

--End assembly, 27 bytes, Errors: 0
```

LISTING 47: MODIFIED LINECHECKER

```
                 *************************************
                 *    MODIFIED LINECHECKER           *
                 *    Sends 3 blanks after each item *
                 *    Copyright (C) 1984             * Merlin
                 *    MicroSPARC, Inc.               * Assembler
                 *    Concord, MA   10742            *
                 *************************************

                 CROUT   EQU  $FD8E      ; Monitor - output a CR
                 PRBLNK  EQU  $F948      ; Monitor - output 3 blanks
                 RDKEY   EQU  $FD0C      ; Monitor - keyboard wait


                         ORG  $ADA3
ADA3: A9 2A              LDA  #$2A       ; Initialize item count

                 * Code here prints out DISC VOLUME NNN

                         ORG  $ADC3
ADC3: 20 8E FD           JSR  CROUT      ; Put out a real CR
ADC6: 20 8E FD           JSR  CROUT      ;   and another

                 * This prints out info for each file and calls LNCHK

                         ORG  $AE2F
AE2F: EA         LNCHK   NOP
AE30: EA                 NOP
AE31: 20 48 F9           JSR  PRBLNK     ; Send 3 blanks after item
AE34: CE 9D B3           DEC  $B39D      ; Decrement item count
AE37: D0 08              BNE  $AE41      ; Branch if count not expired
AE39: 20 0C FD           JSR  RDKEY      ; If expired, wait for keyin
AE3C: A9 2A              LDA  #$2A       ; then restart item count
AE3E: 8D 9D B3           STA  $B39D      ; and save it for future use
AE41: 60         RTRN    RTS             ; Return to caller

--End assembly, 27 bytes, Errors: 0
```

LISTING 48: CATALOG80.ML

```
                 ****************************************
                 *          CATALOG80.ML                *
                 *    BY ROBERT MELTZER                 *
                 *    COPYRIGHT (C) 1984                * Merlin
                 *    MicroSPARC, Inc.                  * Assembler
                 *    Concord, MA   01742               *
                 ****************************************

                 RDKEY    EQU   $FD0C      ; Monitor keybd wait routine
                 COUT     EQU   $FDED      ; Monitor rtn to output (A)
                 CROUT    EQU   $FD8E      ; Monitor rtn to output a CR
                 PRBL2    EQU   $F94A      ; Monitor rtn to send blanks


                          ORG   $ADA3
ADA3: A9 2C               LDA   #$2C       ; Initialize item count


                          ORG   $ADA8
ADA8: A9 13               LDA   #$13       ; Set tab for volume header
ADAA: 85 24               STA   $24        ; and force horiz cursor posn
ADAC: EA                  NOP
ADAD: EA                  NOP

                 * Code here prints out DISC VOLUME NNN

                          ORG   $ADC3
ADC3: 20 8E FD            JSR   CROUT      ; Put out a Carriage Return
ADC6: 20 8E FD            JSR   CROUT      ;   and another

                 * This prints out info for each file and calls LNCHK

                          ORG   $AE2F
AE2F: EA         CLNUP    NOP
AE30: CE 9D B3            DEC   $B39D      ; Decrement item count
AE33: D0 08               BNE   JPTCH      ; If not zero,
AE35: 20 0C FD            JSR   RDKEY      ; If zero, wait for keypress
AE38: A9 2C               LDA   #$2C       ;   then reinitialize the
AE3A: 8D 9D B3            STA   $B39D      ;     item count value
AE3D: A2 02      JPTCH    LDX   #02        ; # blanks for PRBL2
AE3F: 4C DF BC            JMP   $BCDF      ; Jump to the patch area


                          ORG   $BCDF      ; This is the patch area
BCDF: 20 4A F9   PATCH    JSR   PRBL2      ; Output (X) blanks
BCE2: A2 8D               LDX   #$8D       ; Assume we need a CR
BCE4: AD 9D B3            LDA   $B39D      ; Get the count
BCE7: 6A                  ROR              ;   and put its low bit in C
BCE8: 90 02               BCC   *+4        ; If C=0 then CR
BCEA: A2 A0               LDX   #$A0       ; If C=1, blank instead
BCEC: 8A                  TXA
BCED: 20 ED FD            JSR   COUT       ; Whichever it is, put it out
BCF0: 60                  RTS              ;   and return to caller

--End assembly, 51 bytes, Errors: 0
```

Hi-Res Characters

Use this exact replica of the Apple's character set to add text to your Hi-Res graphics. A short Applesoft program shows you how.

by Vinay, Vivek, and Vijay Pai

When working in high resolution graphics on the Apple, the use of the Apple's character set is often required to display text on a Hi-Res screen. Programmers who find that the four lines of text at the bottom of the screen are not adequate often forgo graphics and revert back to an all-text program. With the shape table presented here, you can say good-bye to your Hi-Res dilemmas for good! Hi-Res Characters consists of two parts: a shape table (**Listing 49**) and an Applesoft demonstration program (**Listing 50**).

The shape table in **Listing 49** is an exact replica of the Apple II's character set. All special symbols, numbers, and uppercase letters are included, in the order that they appear in the Apple's character set. Since the Apple's character set and the shape table coincide, Hi-Res characters can easily be drawn.

USING THE SHAPE TABLE

The shape table shown in **Listing 49** begins at \$6000, and has a length of \$313. Applesoft's shape table pointers must be set to the beginning address of the table. \$E8 (232 decimal) holds the low-order byte of the beginning address, and \$E9 (233) holds the high-order byte. Since the shape table begins at \$6000, \$00 (0) is the low-order byte and \$60 (96) is the high-order byte.

From Applesoft, the POKEs 232,0 and 233,96 must be used. From the Monitor, \$E8 must be set to \$00, and \$E9 must be set to \$60. Either the Applesoft approach or the Monitor approach may be used. The shape table may be BLOADed at a different address but the Applesoft shape table pointers must be set to the new value of the beginning address.

DRAWING THE SHAPES

To draw a shape from a program, first determine the ASCII code of the letter to be drawn using the ASC function. Subtract 31 from this value, then draw using the result from the subtraction, i.e., DRAW (ASC (letter) - 31) AT X,Y. To obtain double-width characters, first draw the shape at X,Y. Then draw the same shape at (X+1) or (X-1),Y.

DEMONSTRATION PROGRAM

CHAR.SET.DEMO (**Listing 50**) shows how to test each shape by comparing it to its counterpart in the Apple character set. A subroutine to draw sentences or phrases is also included and explained.

ENTERING THE PROGRAMS

Please refer to Appendix A for help in entering **Listing 49**. If you key it in from the Monitor, save it to disk with the command:

```
BSAVE HI.RES.CHAR.SET,A$6000,L$313
```

To key in CHAR.SET.DEMO, type in the program as shown in **Listing 50** and save it to disk with the command:

```
SAVE CHAR.SET.DEMO
```

LISTING 49: HI.RES.CHAR.SET

```
6000- 40 00 82 00 84 00 8B 00
6008- 91 00 9F 00 AD 00 BA 00
6010- C6 00 C9 00 D4 00 DD 00
6018- EC 00 F3 00 F8 00 FD 00
6020- 00 01 07 01 15 01 1D 01
6028- 28 01 35 01 42 01 4E 01
6030- 5A 01 65 01 71 01 7D 01
6038- 80 01 84 01 8E 01 96 01
6040- A0 01 AA 01 B7 01 C3 01
6048- CF 01 DC 01 E8 01 F4 01
6050- 00 02 0C 02 18 02 22 02
6058- 2A 02 37 02 40 02 4D 02
6060- 5A 02 66 02 70 02 7E 02
6068- 8C 02 97 02 A0 02 AB 02
6070- B6 02 C2 02 CE 02 D8 02
6078- E5 02 F3 02 FB 02 08 03
6080- 0E 03 00 00 B6 04 40 18
6088- 24 04 00 18 24 0D 36 04
6090- 00 83 24 6C 36 FF 16 2D
6098- 25 0C 16 17 FE 24 00 24
60A0- 74 39 3F 17 0E 0D 0E 1E
60A8- 27 1E 77 21 00 18 38 2C
60B0- 56 09 B8 17 17 17 4D 35
60B8- 27 00 20 1C 17 76 1E 76
60C0- 65 1C 8C B1 04 00 20 24
60C8- 00 1B 40 18 09 17 1E 36
60D0- 0E 0E 04 00 40 18 0E 0E
60D8- 36 1E 1E 04 00 24 34 50
60E0- F1 1E 18 1C 96 62 0D 0E
60E8- 1F B4 04 00 2D DF 27 48
60F0- B6 26 00 12 30 1E 04 00
60F8- 1B 2D 2D 04 00 92 04 00
6100- 40 B9 17 17 17 04 00 0C
6108- 0C 1C 3F 1E 36 2E 1E 0E
6110- 2D 0C 24 24 00 24 BC 96
6118- 31 3E 0D 04 00 65 E4 3F
6120- 1E 96 F1 CE 2D 2D 04 00
6128- 25 05 20 3F 3F 96 4A 09
6130- F6 3F 1C 04 00 2A 36 04
6138- 28 07 20 24 17 17 17 2E
6140- 04 00 28 1F 27 2C 2D B5
6148- 32 F6 3F 1C 04 00 39 3F
6150- 2C 60 2D 96 32 1E 3F 1C
6158- 24 00 1A 0C 0C 0C 3C 3F
6160- B7 92 31 04 00 39 E7 2C
6168- 28 75 B6 F6 3F 07 20 04
6170- 00 2D 24 07 38 F7 76 4E
6178- F1 1E 3F 04 00 B0 04 00
6180- B0 F6 04 00 1B 0C 0C 0C
6188- 96 92 1C 1C 04 00 D8 2D
6190- 2D D6 39 3F 27 00 09 07
6198- 38 E0 96 4A 1E 1E 04 00
```

```
61A0- 92 04 20 0C 0C 1C 3F 1E
61A8- 04 00 30 2E 2C 24 1C 3F
61B0- 1E 36 36 0E 2D 25 00 2A
61B8- 25 3C 38 B8 17 36 F5 6E
61C0- 09 24 00 3F 24 2C 2D 15
61C8- F6 0E F6 3F 27 24 00 09
61D0- 40 03 1C 3F 1E 36 36 15
61D8- 2D 0C 04 00 1B 24 2C 2D
61E0- 0E 36 36 17 3F 27 24 00
61E8- 25 40 3F 3F 36 2E 1E 36
61F0- 2D 2D 04 00 19 2D 40 18
61F8- 3F 3F 36 2E 1E 36 04 00
6200- 0A 46 36 3F 3F 20 24 2C
6208- 28 2D 04 00 2D 24 FC 1B
6210- 36 2E 1E 36 4D 21 24 00
6218- 40 18 2B F5 36 36 3E 0D
6220- 04 00 93 73 2D 0C 24 24
6228- 24 00 63 0C 0C DF 33 36
6230- 36 6E 09 07 38 20 00 1B
6238- 24 B4 12 36 2D 2D 04 00
6240- 64 05 30 36 36 FE 1B 24
6248- 24 24 0E 04 00 18 0E 0E
6250- 56 24 24 24 DF 33 36 36
6258- 26 00 1B 24 05 28 75 36
6260- 36 1E 3F 1C 24 00 65 3C
6268- 38 3F 36 2E 1E 36 04 00
6270- 8A 11 1C 07 68 24 3C 38
6278- F7 36 36 0E 25 00 65 3C
6280- 38 37 18 36 2E 0E 0E 0E
6288- DF 23 24 00 75 F6 3F 1C
6290- 44 1C 64 2D 0E 04 00 24
6298- 3C 6F 29 96 DA 36 04 00
62A0- 1B 24 6C 09 36 36 F6 3F
62A8- 1C 24 00 92 0C 0C 24 24
62B0- DF 33 36 AE 04 00 76 0E
62B8- 24 24 24 DF 33 36 36 66
62C0- 04 00 0C 0C FC 1B AE 16
62C8- 17 6E 09 E4 04 00 07 68
62D0- 0C FC 1B B6 4A 36 04 00
62D8- 17 1E 2E 2D 25 40 18 2B
62E0- 20 3F 3F 04 00 1B 24 2C
62E8- 2D F5 1B 36 36 2E 2D DF
62F0- 23 24 00 19 1C 1C 96 49
62F8- 0E 04 00 21 24 3F 6F 09
6300- 36 36 36 27 B4 3B 27 00
6308- B8 17 4D E1 04 00 92 1F
6310- 6D 25 00
```

LISTING 50: CHAR.SET.DEMO

```
10  REM  *********************
20  REM  *    CHAR.SET.DEMO   *
30  REM  *   BY VINAY, VIVEK, *
40  REM  *    AND VIJAY PAI   *
50  REM  * COPYRIGHT (C) 1984 *
60  REM  * BY MICROSPARC, INC *
70  REM  * CONCORD, MA. 01742 *
80  REM  *********************
90  REM
100  PRINT  CHR$ (4);"BLOAD HR.CHAR.SET"
110  HOME : TEXT : HOME : SPEED= 255: NORMAL
120  POKE 232,0: POKE 233,96: REM POKE IN HIGH AND LOW BYTES OF
     SHAPE TABLE ADDRESS.
130  HGR : ROT= 0: SCALE= 1: HCOLOR= 7
140 A$ = "PRESS A KEY TO":B$ = "VIEW NEXT LETTER"
150  GOSUB 160: GOTO 250
160  FOR T = 1 TO  LEN (A$): DRAW ( ASC ( MID$ (A$,T,1)) - 31)
     AT T * 10,20
170  DRAW ( ASC ( MID$ (A$,T,1)) - 31) AT T * 10 + 1,20
180  NEXT
190  FOR T = 1 TO  LEN (B$): DRAW ( ASC ( MID$ (B$,T,1)) - 31)
     AT T * 10,40
200  DRAW ( ASC ( MID$ (B$,T,1)) - 31) AT T * 10 + 1,40
210  NEXT
220  REM THE PROGRAM 'READS' EACH LETTER OF A$ OR B$, AND DRAWS
     ITS ('ASC' VALUE - 31).
230  REM THE SAME LETTER IS BEING DRAWN AGAIN, EXCEPT AT THE
     NEXT PIXEL TO GIVE THE LETTERS THEIR THICKNESS. THIS
     PROCESS MAY BE OMITTED TO ACHIEVE NORMAL WIDTH, APPLE-
     CHARACTER SHAPES.
240  RETURN
250 C$ = "< HI-RES CHARACTER": FOR T = 1 TO  LEN (C$): DRAW (
     ASC ( MID$ (C$,T,1)) - 31) AT (T * 10 + 20),100: DRAW ( ASC
     ( MID$ (C$,T,1)) - 31) AT (T * 10 + 21),100: NEXT
260  VTAB 23: PRINT "** COPYRIGHT 1984 BY MICROSPARC, INC. **";:
     VTAB 21: HTAB 5: PRINT "< APPLE'S CHARACTER (IN TEXT
     WINDOW)"
270  FOR T = 32 TO 95
280  VTAB 21: HTAB 1: PRINT  CHR$ (T)
290  HCOLOR= 7: DRAW T - 31 AT 3,100
300  WAIT  - 16384,128: POKE  - 16368,0
310  HCOLOR= 0: DRAW T - 31 AT 3,100
320  NEXT T
330  HGR : HOME
340 A$ = "TYPE IN UP TO 20 CHARACTERS":B$ = "(NO CONTROL
     CHARACTERS)"
350  HCOLOR= 7: GOSUB 160
360  VTAB 21: HTAB 1: INPUT ":";A$
370  IF  LEN (A$) < 1 OR  LEN (A$) > 20 THEN  PRINT  CHR$ (7):
     GOTO 360: REM CHECK LENGTH OF INPUT
380 C$ = "--------------------":B$ =  LEFT$ (C$, LEN (A$)): REM
     20 DASHES
```

```
390  FOR T = 1 TO  LEN (A$): IF  ASC ( MID$ (A$,T,1)) < 32 OR
     ASC ( MID$ (A$,T,1)) > 95 THEN  PRINT  CHR$ (7): GOTO 360:
     REM CHECK FOR ILLEGAL CHARACTERS
400  NEXT T
410  HGR : HCOLOR= 7: GOSUB 160: REM CLEAR SCREEN, AND DRAW WHAT
     USER HAS TYPED IN.
     410  HGR : HCOLOR= 7: GOSUB 160: REM CLEAR SCREEN, AND DRAW
     WHAT USER HAS TYPED IN.
420  VTAB 22: PRINT "PRESS ANY KEY TO QUIT";: GET K$: TEXT :
     HOME : END
```

Applesoft Windows

The Apple's text display is a window that you can control from within your own programs. These short programs show you how it's done in both Applesoft and machine language.

by Michael A. Seeds

Your Apple Monitor contains a window, and looking through that window can solve a few bothersome programming problems. For example, I like to jump around when I am editing a program, and sometimes I need to copy parts of one section into another section. I've often wished I could run two video monitors side by side — one to display the program and one to display my working area. Another problem is displaying short messages without disturbing text already on the screen.

CHANGING THE WINDOW'S BOUNDARIES

We can solve problems like these using the Apple text window. Normally the window is set to fill the entire screen, but you can change the boundaries of the text window by POKEing locations 32-35. A POKE 32,10, for instance, sets the left margin of the text window to the tenth space from the left. You can try this in immediate mode, if you like. Try these locations:

Location	Function	Limits
32	Left edge	0-39
33	Width of window	1-40
34	Top edge	0-23
35	Bottom edge	0-23

(For more details, see page 31 of the *Apple II Reference Manual* or Appendix F of the *Applesoft BASIC Programmer's Reference Manual*.)

When you change the text window, your Apple uses the new area and ignores anything outside it. The HOME command clears just the window and places the cursor at the upper left corner of the window. If you list a program, the text in the window will scroll as usual, but text outside the window will be left untouched. HTAB will move the cursor relative to the newly-defined left edge, but VTAB will allow you to move the cursor above or below the existing text window. Go ahead and try a few POKEs. No matter how much you mess up the window boundaries, you can restore the window to full screen with a TEXT command.

Using these same locations, programs can very easily control the window boundaries. The following two programs present two possibilities. The first, BIWIND (**Listing 51**), gives you two work areas on the screen for program development and testing. The second, WINDER (**Listing 52**), demonstrates a technique for creating Macintosh-like "dialog" windows.

BIWIND

BIWIND is a short machine language program that allows you to divide the screen into top and bottom sections. You can work in either half without disturbing the other. With BIWIND installed, you can enter the top half of the screen by moving the cursor to the bottom half and typing CALL 771. When you press <RETURN> the screen will divide in half and you will be working in the top half. You can LIST, EDIT, and RUN programs here without disturbing the text below (unless, of course, your program alters locations 32-35 or uses a TEXT command).

To enter the bottom half of the screen, move the cursor to the top half and type CALL 794. Press <RETURN> and the bottom window will open for your use. To return to the full screen, just type TEXT.

BIWIND is a simple program, and, because of the way it remembers the last cursor position, it can get confused if you try to open the half of the screen in which you are already working. If that happens, try typing HOME. If things get hopelessly confused, just type TEXT and you will be back to a full screen.

I find BIWIND especially useful for editing my programs. For example, I can jump to the top screen to list one segment of the program, and then jump back to the bottom screen to edit a related part of the program.

The program will always divide the screen into two equal areas unless you specify otherwise. To divide the screen at the *n*th line, just POKE 770, *n*. The next time you open one of the work areas, the new dividing line will be in effect.

If you use these subroutines in your own programs, notice that the variables they use all begin with a W. Avoid using W variables in the rest of your program so that the window subroutines won't interfere with them. Notice, also, that your main program must dimension WS$(24).

ENTERING BIWIND

Please refer to Appendix A for help in entering BIWIND. If you key it in from the Monitor, save it to disk with the command:

BSAVE BIWIND,A$300,L$45

HOW BIWIND WORKS

This machine language program is really two separate routines. The first opens the top of the screen, while the second opens the bottom of the screen. Let's look at the first routine.

When we CALL 771, the program saves the present location of the cursor for use when we flip back to the bottom half of the screen, and then it loads in the last location of the cursor in the top half of the screen. The Apple always stores the current cursor location in $25. Next, the program sets the top of the working area to zero and it sets the bottom line to the contents of $302. Finally, it calls the subroutine at $334 to draw a line of equal signs dividing the two screen areas.

The second routine opens the bottom of the screen. This routine is very similar to the first. The major difference is that it sets the top of the screen to the contents of $302 plus 1. This prevents it from overwriting the dividing line of equal signs.

WINDER

The program WINDER (**Listing 52**) also uses the text window, but for a different purpose. The subroutines starting at **line 390** can be used in any Applesoft program to open a small window in a text screen display. I use this to display messages to the user. The subroutine at **line 560** closes the temporary message window and restores the original data. The first part of the program in **Listing 52** is just a demonstration of the methods.

To open a window, the main program must set the quantities WL, WT, WW, and WB. These are the four numbers to be POKEd into locations 32-35, and they define the location and size of the window. To allow room for a border, WW and WB must be greater than two. Of course, the boundaries of the window must not go beyond the boundaries of the video screen.

ENTERING WINDER

To key in WINDER, type in the Applesoft program shown in **Listing 52** and save it on disk with the command:

SAVE WINDER

LISTING 51: BIWIND

```
                   ;
                   ;
                   ; BIWIND
                   ; BY MIKE SEEDS
                   ; COPYRIGHT (C) 1985
                   ; BY MICROSPARC, INC.
                   ; CONCORD, MA 01742
                   ;
                   ; MICROSPARC ASSEMBLER
                   ;
                            ORG $300
                   VCURS    EQU $25       ;VERTICAL CURSOR POSITION
                   BOTSCR   EQU $23       ;BOTTOM OF TEXT WINDOW
                   TOPSCR   EQU $22       ;TOP EDGE OF WINDOW
0300  0B           TOP      DFC 11        ;TOP CURSOR POSITION
0301  17           BOT      DFC 23        ;BOTTOM CURSOR POSITION
0302  0C           LINE     DFC 12        ;DIVIDING LINE SET AT 12
0303  A5 25                 LDA VCURS     ;*** OPEN TOP ***
0305  8D 01 03              STA BOT      ;SAVE BOTTOM CURSOR POSITION
0308  AD 00 03              LDA TOP       ;
030B  85 25                 STA VCURS     ;SET TOP CURSOR POSITION
030D  A9 00                 LDA #$0
030F  85 22                 STA TOPSCR    ;SET TOP OF AREA
0311  AD 02 03              LDA LINE      ;GET DIVIDING LINE
0314  85 23                 STA BOTSCR    ;SET BOTTOM OF AREA
0316  20 34 03              JSR DIVIDE    ;DRAW DIVIDING LINE
0319  60                    RTS           ;END OF OPEN TOP ROUTINE
                   ;
031A  A5 25                 LDA VCURS     ;*** OPEN BOTTOM ***
031C  8D 00 03              STA TOP       ;SAVE TOP CURSOR POSITION
031F  AD 01 03              LDA BOT
0322  85 25                 STA VCURS     ;SET BOTTOM CURSOR POSITION
0324  AD 02 03              LDA LINE      ;GET DIVIDING LINE
0327  18                    CLC
0328  69 01                 ADC #1        ;ADD ONE
032A  85 22                 STA TOPSCR    ;SET TOP OF AREA
032C  A9 18                 LDA #$18      ;DECIMAL 24
032E  85 23                 STA BOTSCR    ;SET BOTTOM OF AREA
0330  20 34 03              JSR DIVIDE    ;DRAW DIVIDING LINE
0333  60                    RTS           ;END OF OPEN BOTTOM ROUTINE
                   ;
0334  AD 02 03     DIVIDE   LDA LINE      ;GET LINE POSITION
0337  20 24 FC              JSR $FC24     ;VTAB TO DIVIDING LINE
033A  A2 27                 LDX #39       ;PRINT 39 SYMBOLS
033C  A9 BD                 LDA #$BD      ;= SIGN
033E  20 F0 FD     OUT      JSR $FDF0     ;PRINT A SYMBOL
```

```
0341  CA        DEX
0342  D0 FA     BNE OUT      ;LAST SYMBOL?
0344  60        RTS          ;DONE

000  ERRORS

0300  HEX START OF OBJECT
0344  HEX END OF OBJECT
0045  HEX LENGTH OF OBJECT
95C3  HEX END OF SYMBOLS
```

LISTING 52: WINDER

```
10  REM  ************************
20  REM  *        WINDER        *
30  REM  *     BY MIKE SEEDS    *
40  REM  *  COPYRIGHT (C) 1985  *
50  REM  *  BY MICROSPARC, INC  *
60  REM  *  CONCORD, MA. 01742  *
70  REM  ************************
80  DIM WS$(24)
90  HOME : PRINT : PRINT  TAB( 9)"FOR WHOM THE BELL BONGS"
100  PRINT : PRINT  TAB( 15)"BY A. MONKEY": PRINT : PRINT
110  FOR J = 1 TO 8
120  FOR K = 1 TO 40: PRINT  CHR$ (64 + 26 *  RND (1));: NEXT K:
     PRINT
130  NEXT J
140  VTAB 23: PRINT "PRESS ANY KEY TO HALT."
150 WL = 12:WT = 10:WW = 10:WB = 5: GOSUB 390: REM  OPEN WINDOW
160  PRINT : PRINT "GOT IT?"
170  GOSUB 350: REM  DELAY
180  GOSUB 560: REM  CLOSE WINDOW
190  IF  PEEK (49152) > 128 THEN  TEXT : HOME : END
200 WL = 5:WT = 1:WW = 25:WB = 7
210  GOSUB 390: REM  OPEN WINDOW
220  VTAB WT + 2: HTAB 4: PRINT "NOTICE THE TEXT IS": HTAB 4:
     PRINT "RESTORED CORRECTLY."
230  GOSUB 350: REM  DELAY
240  GOSUB 560: REM   CLOSE WINDOW
250  IF  PEEK (49152) > 128 THEN  TEXT : HOME : END
260 WT = 10:WB = 10: GOSUB 390
270  FOR J = 1 TO 25: PRINT "  ";J,J * J: NEXT J
280  PRINT : PRINT "SCROLLING IS AUTOMATIC"
290  GOSUB 350: GOSUB 560
300  IF  PEEK (49152) > 128 THEN  TEXT : HOME : END
310  GOTO 150
320  REM  ==========
330  REM    DELAY
340  REM  ==========
350  FOR J = 1 TO 1500: NEXT : RETURN
360  REM  ===================
370  REM   SUBROUTINE WINDOW
380  REM  ===================
```

```
390 WA = 1024 + 128 * (WT - 1 - 8 *  INT ((WT - 1) / 8)) + 40 *
    INT (WT / 8.5)
400 WS = WA
410  FOR WJ = WT TO WT + WB - 1:WS$(WJ) = ""
420  FOR WK = 1 TO WW:WS$(WJ) = WS$(WJ) +  CHR$ ( PEEK (WA + WL
    + WK - 1)): NEXT WK
430  POKE WA + WL,32: POKE WA + WL + WW - 1,32
440 WA = WA + 128: IF WA = 2088 THEN WA = 1104
450  IF WA = 2048 THEN WA = 1064
460  NEXT WJ
470  FOR WJ = 1 TO WW: POKE WS + WL + WJ - 1,32: POKE WA - 128 +
     984 * (WA = 1064 OR WA = 1104) + WL + WJ - 1,32: NEXT WJ
480  REM  SET TEXT SCREEN
490  POKE 32,WL + 1: POKE 33,WW - 2
500  POKE 34,WT: POKE 35,WT + WB - 2
510  HOME
520  RETURN
530  REM  ================
540  REM  SUBROUTINE CLOSE
550  REM  ================
560  POKE 32,0: POKE 33,40
570  POKE 34,0: POKE 35,24
580  FOR WJ = WT TO WT + WB - 1: VTAB WJ: HTAB WL + 1: PRINT
     WS$(WJ): NEXT WJ
590  RETURN
```

80-Column Screen Dump

Use this handy Applesoft routine to send the contents of the 80-column display of your IIe , IIc or IIGS to a printer

by A. R. Clayton

If you program the Apple IIe or IIc, you may have wondered whether you can read the 80-column screen from Applesoft. It is possible, and this program demonstrates how the 80-column display can be read a line at a time from within an Applesoft program, and sent to a printer.

USING THE PROGRAM

To use the program, simply include the subroutine shown in **lines 100-290** of **Listing 53** in your own program. Whenever you want the contents of the 80-column screen sent to the printer, use a GOSUB 100 statement. **Listing 53** demonstrates the use of the program by first turning on the 80-column card, issuing a CATALOG command to fill the screen, and finally, using a GOSUB 100 to send the screen contents to the printer.

There are several methods of including this routine in your own program. The most obvious is to simply load the routine, and then begin writing your program at **line 300**. (Other options are to use the merge function in the RENUMBER program on the DOS 3.3 System Master disk or to use one of the many programming utilities available that perform this function.) A text file containing the code may be created and then EXECed into your program as described in the *DOS Manual* on p. 76.

ENTERING THE PROGRAM

To key in the program, enter the Applesoft program as shown in **Listing 53** and save it to disk with the command:

SAVE SCREEN.DUMP80

You must have a IIe with an 80-column card installed, or a IIc or IIGS, and your printer card must be in slot 1. If your printer card is in another slot, change the statement in **line 100** to the appropriate slot. Also, IIc and IIGS owners should change the first statement in **line 200** to:

```
PRINT L$
```

HOW THE PROGRAM WORKS

We assume that the 80-column display is active and that there is something on the screen upon entry to the subroutine. Three FOR-NEXT loops then begin to execute, reading each line across the screen until 80 characters have been read in. The 80-column screen is simply two 40-column screens that share the same memory block.

Starting at $400 (1024 decimal), every other character is stored on the auxiliary screen. The first character of each line is stored on the alternate screen. To turn on the auxiliary screen and read the first character, POKE 49237,0 and then PEEK(1024). This will return the ASCII value of the first character on the screen. The next character would be read in by turning off the auxiliary screen with the commands POKE 49236,0 and the PEEK(1024). For example:

Memory location	1024	1024	1025	1025
Character on screen	A	B	C	D
Memory	AUX	MAIN	AUX	MAIN

For a more detailed explanation, see p. 29 of the *Apple IIe Reference Manual.*

The following functions are performed by the routine: **Lines 100-110** turn on the printer and turn off the screen display to prevent characters from being sent to the screen. **Lines 120-140** set up three FOR-NEXT loops that step through each memory location used by the screen display.

Lines 150 and 170 toggle the soft switch between auxiliary and main memory. The subroutine at **line 260** removes control characters by testing their ASCII values. If the value is less than 31, a space character is put in the place of the control character.

Line 200 sends L$ to the printer. This string was built in the subroutine that starts at **line 260**. When all 80 characters have been read, it is nulled and used again. Finally, **lines 230 and 240** turn off the printer and turn on the 80-column card.

MODIFICATIONS

Because both the 80-column display and the printer interface are treated as peripherals, the PR#0 used to switch off output to the printer does not transfer output back to the 80-column card in slot 3. Unfortunately, this means that a new PR#3 must be issued, which will clear the screen.

If it is important that your program return to 80-column display with the original display intact, you may want to modify the program. First, dimension a string array large enough to hold each line of the 80-column screen. Then use this array, rather than the variable L$, to build each line before sending it to the printer. After issuing the PR#0 and PR#3 to redirect output to the 80-column card, use a FOR-NEXT loop to reprint the screen from the string array.

LISTING 53: SCREEN.DUMP80

```
10  REM  ***********************
20  REM  *     SCREEN.DUMP80     *
30  REM  *     BY A.R.CLAYTON    *
40  REM  *   COPYRIGHT (C) 1985  *
50  REM  *   BY MICROSPARC, INC  *
60  REM  *   CONCORD, MA. 01742  *
70  REM  ***********************
80  PRINT  CHR$ (4);"PR# 3": PRINT  CHR$ (4)"CATALOG": REM  TURN
     ON 80 COL CARD AND PUT SOMETHING ON THE SCREEN
90  GOSUB 100: END
100  PRINT  CHR$ (4);"PR# 1": REM  TURN ON PRINTER
110  PRINT  CHR$ (9);"80N": REM  TURN OFF SCREEN
120  FOR A = 1024 TO 1104 STEP 40: REM  SEE MEM MAP 80 COL TEXT
     PAGE 32 IIe REF
130  FOR I = A TO A + 896 STEP 128
140  FOR X = I TO I + 39
150  POKE 49237,0: REM  TURN PAGE 2 ON AUX MEM
160  GOSUB 260
170  POKE 49236,0: REM   TURN PAGE 2 OFF
180  GOSUB 260
190  NEXT X
200  PRINT L$:L$ = "": REM   PRINT L$ AND THEN NULL L$
210  NEXT I
220  NEXT A
230  PRINT  CHR$ (4);"PR# 0": REM  TURN PRINTER OFF
240  PRINT  CHR$ (4);"PR# 3": REM  TURN 80 COL BACK ON
250  RETURN
```

```
260 L =  PEEK (X)
270 L = L * (L < 255) * (L > 31): IF  NOT L THEN L = 32
280 L$ = L$ +  CHR$ (L)
290  RETURN : REM  THIS SUB REMOVES CONTROL CHARACTERS
```

Catalog Plus

Modify the CATALOG command to display only the files you want to see. This patch to DOS 3.3 allows you to catalog by file type or by the first character of the file name.

by Bryan Costales

The CATALOG function is the DOS 3.3 function that is most often tinkered with and patched. Whether designed to skip the pause during a listing or to print JOE'S DISK instead of DISK VOLUME, patches to the CATALOG all seem to be aimed at fixing a less-than-optimum routine.

I have always had problems finding a particular text file among 50 or so A, I, and B type files, not to mention getting a separate listing of my source code files when both my source and object files are binary type files. Why not, I thought, patch the CATALOG code to allow it an argument or two? Why not alter the CATALOG to list only text files or only files that start with the letter S, for that matter?

CATALOG.PLUS (**Listing 54**) solves these problems by allowing arguments in the CATALOG command line. In designing the program, I decided to adhere to the following restrictions:

1. The program does not use any of the so-called unused space in DOS.
2. It does not change the locations of any called subroutines (e.g., $AE2F, which prints a carriage return and produces a pause.)
3. It does not change the plain CATALOG command itself, as many programs call that function by name.
4. It does not change page 3 access, since programs like FID use it to produce a CATALOG listing.

USING CATALOG PLUS

After you BRUN CATLOG.PLUS, several options are available. The CATALOG command still provides the full list of all the files on the disk. You have the option of providing either of two arguments to limit your catalog to files of a certain type, files that begin with a certain character, or both.

In the statement:

```
CATALOGfiletypefilename
```

filetype is the character representing the file type, such as I, B or A. A space following the CATALOG command causes all file types to be selected. The *filename* field is the first letter of a file name. If the *filename* field is omitted, all file names may be selected. Examples of the possible combinations are shown in **Figure 1**. Pressing any key will stop or restart the listing.

FIGURE 1: *Examples of CATALOG Functions*

```
]CATALOG
*B 020 FID
 T 005 LETTER.JOHN 12/81
 T 004 EXEC LISTER
 A 013 LISTER

]CATALOGB
*B 020 FID
```

```
]CATALOGL
 T 005 LETTER.JOHN 12/81
 A 013 LISTER

]CATALOGTL
 T 005 LETTER.JOHN 12/81
```

Functions Lost or Changed

If you enter CATALOGD1, the function will no longer give the full catalog of the disk in drive 1. You must now delimit the drive number with a comma. For instance, you would type CATALOG,D1 or CATALOG B,D1. Also, the words DISK VOLUME are no longer printed, nor is the volume number displayed.

ENTERING THE PROGRAM

Please refer to Appendix A for help in entering this program. If you key it in from the Monitor, start at $6000. The program should be saved to disk with the command:

BSAVE CATALOG.PLUS,A$6000,L$14C

When it is BRUN, CATALOG.PLUS rewrites the catalog code within DOS. The CATALOG.PLUS code will be a permanent part of the DOS of any slave disk initialized from this disk.

HOW IT WORKS

Whenever DOS finds a valid DOS command, it first checks its Command Parameter Table at $A909 to see if that command requires a file name. By altering the byte at $A929 from a $40 to a $60, DOS is tricked into thinking that CATALOG requires a file name. DOS then obligingly finds the file name and places it in the primary name buffer at $AA75.

To ensure that CATALOG will always have a file name after it, we substitute the new name CATALO into the Command Name Table at $A884. Just to be on the safe side, the whole table is rewritten from scratch. If you had DOIT for RUN and CAT for CATALOG, you will now find all the standard terms restored along with CATALO.

Now, if CATALOG is entered, DOS saves the file name G in the primary buffer. If CATALOGAT is entered, it saves the file name GAT. Next it compares the ASCII character for each file's type to the second character in the buffer, and compares the first letter of each file name to the third character in the buffer. Files are accepted for display when a match is found.

The File Manager lets you exit from the Catalog function. When the File Manager finds a name in the buffer, it assumes that a file has been opened. If it were to remain open, you would soon receive a NO BUFFERS AVAILABLE error. To prevent this, a JSR $A764 locates the opened file, places a $00 there, and closes it.

Finally, a JSR $A095 clears the primary file name buffer, so any page 3 callers will get a bad listing only on the first try. (You will find that FID will catalog only Integer BASIC files that start with the letter D. This is a first-time-only bug. Thereafter, all works as it should.)

MODIFYING THE PROGRAM

Before you attempt to modify this routine, a few words of caution are in order:

1. CATALOG.PLUS utilizes exactly the same space as the old Catalog. To add anything, you must first remove something from the program.

2. The CROUT subroutine was not moved from location $AE2F because other patches to DOS call it at that location (see, for instance, "Free Space," Golding/Pump, *Call -A.P.P.L.E.*, April 1982).
3. I placed the high/low bytes of the file length in locations $44/$45 before calling PRINLEN at $AE42. Although PRINLEN only utilizes $44, you may want to modify it to allow lengths over 244 (see "Disassembly Lines," *Nibble* Vol.3/No.2).

LISTING 54: CATALOG.PLUS

```
;
;
;     CATALOG.PLUS
;
;  BY BRYAN COSTALES
;  COPYRIGHT (C) 1985
;  BY MICROSPARC, INC.
;  CONCORD, MA 01742
;
         USE MACLIB,D1    ;MACRO LIBRARY
         MUL              ; CALLED IN FOR DCI DIRECTIVE
         UEN
;
         ORG $6000
;
;   *** ZERO PAGE LOCATIONS
;
BUFFPTR  EQU $40
LENCALO  EQU $44
;
;   *** DOS LOCATIONS
;
CLRBUFFS EQU $A095              ; Clear buffers.
FINDBUFF EQU $A764              ; Locate open buffer.
KEYTABLE EQU $A884              ; DOS cmd name table.
PARAM    EQU $A929              ; Catalog parameter H.
VOLSAVE  EQU $AA66              ; Last volume number.
SOUGHT   EQU $AA77              ; File/name sought.
INITFM   EQU $ABDC              ; Init fmgr workarea.
CATLOG   EQU $AD98              ; Start of Catalog.
PRINLEN  EQU $AE42              ; Print 2hex as 3dec.
INVTOC   EQU $AFF7              ; Read/write the VTOC.
GETSEC   EQU $B011              ; Read a dir sector.
ADVINDX  EQU $B230              ; Advance to next file.
FMGRXT   EQU $B37F              ; Exit through filemgr.
DIRDEX   EQU $B39C              ; Directory index.
TYPNAM   EQU $B3A7              ; File type letters.
DIRTRK   EQU $B4C6              ; Track of T/S list.
FITYPE   EQU $B4C8              ; File type & lock bit.
FISIZE   EQU $B4E7              ; File size in sectors.
VOLNUM   EQU $B5F9              ; Vol number current.
;
;   *** MONITOR LOCATIONS
;
KEYBRD   EQU $C000
KEYSTRB  EQU $C010
COUT     EQU $FDED
```

```
                   KEYIN    EQU $FD0C
                   ;
                   ;   *** CONSTANTS
                   ;
                   ZERO     EQU $00
                   CR       EQU $8D
                   SPACE    EQU $A0
                   ZROCMP   EQU $FF            ; Zero's Complement.
                   ;
                   ; RELOCATION ROUTINE:
                   ;
6000  A2 A9                 LDX #1+END-START
6002  BD A4 60     RLOOP    LDA START,X
6005  9D 98 AD              STA CATLOG,X
6008  CA                    DEX
6009  E0 FF                 CPX #ZROCMP
600B  D0 F5                 BNE RLOOP
                   ;
                   ; CHANGE  CATALOG TO CATALO
                   ;
600D  A2 00                 LDX #ZERO
600F  BD 20 60     CLOOP    LDA KEYWORDS,X
6012  F0 06                 BEQ PARAM0
6014  9D 84 A8              STA KEYTABLE,X
6017  E8                    INX
6018  D0 F5                 BNE CLOOP
                   ;
                   ; CHANGE CATALOG PARAMATER TO ACCEPT FILENAME
                   ;
601A  A9 60        PARAM0   LDA #$60
601C  8D 29 A9              STA PARAM
601F  60                    RTS
                   ;
6020  49           KEYWORDS DCI "INIT"         ; Hi bit set.
6021  4E
6022  49
6023  D4
6024  4C                    DCI "LOAD"         ; on last letter.
6025  4F
6026  41
6027  C4
6028  53                    DCI "SAVE"
6029  41
602A  56
602B  C5
602C  52                    DCI "RUN"
602D  55
602E  CE
602F  43                    DCI "CHAIN"
6030  48
6031  41
6032  49
6033  CE
6034  44                    DCI "DELETE"
6035  45
6036  4C
6037  45
6038  54
```

```
85  6039  C5
86  603A  4C            DCI "LOCK"
86  603B  4F
86  603C  43
86  603D  CB
87  603E  55            DCI "UNLOCK"
87  603F  4E
87  6040  4C
87  6041  4F
87  6042  43
87  6043  CB
88  6044  43            DCI "CLOSE"
88  6045  4C
88  6046  4F
88  6047  53
88  6048  C5
89  6049  52            DCI "READ"
89  604A  45
89  604B  41
89  604C  C4
90  604D  45            DCI "EXEC"
90  604E  58
90  604F  45
90  6050  C3
91  6051  57            DCI "WRITE"
91  6052  52
91  6053  49
91  6054  54
91  6055  C5
92  6056  50            DCI "POSITION"
92  6057  4F
92  6058  53
92  6059  49
92  605A  54
92  605B  49
92  605C  4F
92  605D  CE
93  605E  4F            DCI "OPEN"
93  605F  50
93  6060  45
93  6061  CE
94  6062  41            DCI "APPEND"
94  6063  50
94  6064  50
94  6065  45
94  6066  4E
94  6067  C4
95  6068  52            DCI "RENAME"
95  6069  45
95  606A  4E
95  606B  41
95  606C  4D
95  606D  C5
96  606E  43            DCI "CATALO"
96  606F  41
96  6070  54
96  6071  41
96  6072  4C
```

```
6073  CF
6074  4D                    DCI "MON"
6075  4F
6076  CE
6077  4E                    DCI "NOMON"
6078  4F
6079  4D
607A  4F
607B  CE
607C  50                    DCI "PR#"
607D  52
607E  A3
607F  49                    DCI "IN#"
6080  4E
6081  A3
6082  4D                    DCI "MAXFILES"
6083  41
6084  58
6085  46
6086  49
6087  4C
6088  45
6089  D3
608A  46                    DCI "FP"
608B  D0
608C  49                    DCI "INT"
608D  4E
608E  D4
608F  42                    DCI "BSAVE"
6090  53
6091  41
6092  56
6093  C5
6094  42                    DCI "BLOAD"
6095  4C
6096  4F
6097  41
6098  C4
6099  42                    DCI "BRUN"
609A  52
609B  55
609C  CE
609D  56                    DCI "VERIFY"
609E  45
609F  52
60A0  49
60A1  46
60A2  D9
60A3  00                    HEX 00
                  ;
                  START
60A4  20 DC AB              JSR INITFM          ; Init fmgr workspc.
60A7  A9 FF                 LDA #ZROCMP
60A9  8D F9 B5              STA VOLNUM          ; Set vol#=0 comp.
60AC  20 F7 AF              JSR INVTOC          ; Read the VTOC.
                  ;
60AF  20 2F AE              JSR CROUT          ; or $BA69 w/Free Space.
                  ;
```

```
60B2  18               CLC                ; For 1st sector.
60B3  20 11 B0 NEXTSEC  JSR GETSEC         ; Get next sector.
60B6  B0 75             BCS ALLDONE        ; Done? Yes, exit.
60B8  A2 00             LDX #ZERO          ; For start of sector.
60BA  8E 9C B3 CATLOOP  STX DIRDEX         ; Save current loc.
60BD  BD C6 B4          LDA DIRTRK,X       ; Get track number.
60C0  F0 6B             BEQ ALLDONE        ; Zero? Yes, exit.
60C2  30 62             BMI TESTDONE       ; Deleted? Yes, skip.
60C4  BD C8 B4          LDA FITYPE,X       ; Get the file type.
60C7  08                PHP                ; Save N-flag status.
60C8  29 7F             AND #$7F           ; and strip lock bit.
60CA  A0 07             LDY #$07
60CC  0A                ASL
60CD  0A       TYPELOOP ASL                ; Shift on-bit to get
60CE  B0 03             BCS GOTTYPE        ; offset to type with
60D0  88                DEY                ; offset in y-reg.
60D1  D0 FA             BNE TYPELOOP
60D3  B9 A7 B3 GOTTYPE  LDA TYPNAM,Y       ; Get Character and
60D6  A8                TAY                ; save it.
60D7  CD 76 AA          CMP SOUGHT-1       ; File type sought?
60DA  F0 07             BEQ CHKLETR
60DC  A9 A0             LDA #SPACE         ; No, a space?
60DE  CD 76 AA          CMP SOUGHT-1
60E1  D0 43             BNE TESTDONE       ; No, skip.
60E3  AD 77 AA CHKLETR  LDA SOUGHT         ; Yes, name sought?
60E6  DD C9 B4          CMP DIRTRK+3,X
60E9  F0 04             BEQ CHKLOCK        ; No, a space?
60EB  C9 A0             CMP #SPACE
60ED  D0 37             BNE TESTDONE       ; No, skip.
60EF  A9 A0    CHKLOCK  LDA #SPACE         ; Yes, locked?
60F1  28                PLP
60F2  10 02             BPL NOLOCK         ; No, print space.
60F4  A9 AA             LDA #"*            ; Yes, print "*".
60F6  20 ED FD NOLOCK   JSR COUT
60F9  98                TYA                ; Restore type letter.
60FA  20 ED FD          JSR COUT           ; and print it.
60FD  A9 A0             LDA #SPACE
60FF  20 ED FD          JSR COUT           ; Print a space.
6102  BD E7 B4          LDA FISIZE,X
6105  85 44             STA LENCAL0        ; Print length.
6107  BD E8 B4          LDA FISIZE+1,X
610A  85 45             STA LENCAL0+1
610C  20 42 AE          JSR PRINLEN
610F  A9 A0             LDA #SPACE         ; Print a space.
6111  20 ED FD          JSR COUT
6114  E8                INX
6115  E8                INX
6116  E8                INX
6117  A0 1D             LDY #$1D           ; Y-reg for 30 letters.
6119  BD C6 B4 NAMELOOP LDA DIRTRK,X
611C  20 ED FD          JSR COUT           ; Print file name.
611F  E8                INX
6120  88                DEY
6121  10 F6             BPL NAMELOOP
6123  20 2F AE          JSR CROUT
6126  20 30 B2 TESTDONE JSR ADVINDX        ; Get next file.
6129  90 8F             BCC CATLOOP        ; More? Yes, again.
612B  B0 86             BCS NEXTSEC        ; No, next sector.
```

```
                    ;
612D  20 64 A7  ALLDONE  JSR FINDBUFF       ; Find opened file.
6130  A2 00              LDX #ZERO
6132  8A                 TXA
6133  81 40              STA (BUFFPTR,X)    ; and close it.
6135  20 95 A0           JSR CLRBUFFS       ; Clear buffers.
6138  4C 7F B3           JMP FMGRXT         ; Exit through filemgr.
                ;
                CROUT    EQU $AE2F
613B  A9 8D              LDA #CR            ; Carriage return.
613D  20 ED FD           JSR COUT
6140  AD 00 C0           LDA KEYBRD         ; Key pressed?
6143  10 06              BPL CROUT0
6145  8D 10 C0           STA KEYSTRB        ; Yes, clear strobe,
6148  20 0C FD           JSR KEYIN          ; and pause.
614B  60        CROUT0   RTS                ; No, return.
                ;
                END

000   ERRORS

6000  HEX START OF OBJECT
614B  HEX END OF OBJECT
014C  HEX LENGTH OF OBJECT
9007  HEX END OF SYMBOLS
```

/RAM — A Free RAM Disk for ProDOS Users

Discover one of Apple's best kept secrets. If you own a IIc or IIGS, or IIe with an extended 80-column card, ProDOS automatically gives you a RAM disk when you boot up.

by Aaron Messing

If you own an Apple IIc or IIGS, or a IIe with the extended 80-column card and ProDOS and don't know about /RAM — get ready for a pleasant surprise. It isn't often that a manufacturer includes a valuable feature in a system and forgets to mention the fact. However, this seems to be the case for /RAM. Except for one brief reference in *BASIC Programming in ProDOS*, Apple has ignored /RAM in their consumer-oriented documentation.

/RAM is the title of a ProDOS volume placed in the additional memory of the extended 80-column card when ProDOS is booted. The process is totally automatic. The resulting RAM disk simulates a disk in a drive connected to the slot containing your extended 80-column card. As with other RAM disks, data manipulation is almost instantaneous, the mechanical noise of a drive is missing, and any information stored in /RAM is lost when power to the system is cut off.

You can obtain information about your system by consulting the ProDOS Filer Utilities on the ProDOS User's Disk. Select Filer Utilities when the main menu appears. Then select Volume Commands from the Filer menu. On the next menu, select List Volumes. A report will appear on the screen that gives the location of /RAM, if it exists, and of other peripherals in your system. /RAM shows up in slot 3, drive 2.

/RAM contains 128 blocks, of which 120 are free. This compares with 273 free blocks on a formatted disk. The size of /RAM notwithstanding, only 12 names will be accepted in its directory.

COMMANDS

Immediate mode commands used regularly in the course of manipulating programs on disk (SAVE, LOAD, RUN, DELETE, LOCK, UNLOCK, CATALOG, etc.) work fine with /RAM. Commands can be formed using the name /RAM or the slot and drive numbers. For example, to run a program enter:

```
RUN /RAM/program name
```

or

```
RUN program name,S3,D2
```

Although both commands will run your program, the second form, which specifies the slot and drive numbers, changes the way the system responds. It allows you to give commands to the system without typing in the slot and drive numbers again. The disk operating system will return to the same slot and drive to complete your commands until you specify different locations.

You can select how you want your system to respond for subsequent commands. If you are busy calling and storing files in /RAM, use a command with slot and drive numbers or change the default prefix with the command PREFIX /RAM. This procedure eliminates typing the same slot and drive information with each new command. To access material within /RAM only occasionally, use the volume title (/RAM),and work patterns with another disk drive will remain undisturbed.

The entire /RAM volume can be erased using the Format a Volume command of the Filer Utilities. If you want, the name /RAM can be changed.

Since there is less space on /RAM than an ordinary disk, /RAM cannot be used with the Copy a Volume command. The mass transfer of files is handles efficiently by the Copy Files command. This command has two wildcard features. The path names requested in the data entry form can be entered as follows. To copy files from the RAM disk to another formatted disk:

Copy Pathname (/RAM/=)
To Pathname (*/your volume name/*=)

To copy from your disk to the RAM disk:

Copy Pathname (*/your volume name/*?)
To Pathname (/RAM/?)

The equal sign and the question mark are the wildcard symbols used to replace the names of the file. Either symbol may be used to transfer files in both directions. However, the same symbol must be used to specify file names for the source and the destination. The program reads the catalog of the source disk. As it copies each file, the program provides information on your screen. There is no need to type each file name. If the equal sign is used, the copying process is continuous for all files. If the question mark is used, you are asked to confirm each choice. Use of wildcard symbols speeds up deletions and alterations of write-protection.

DIRECTORY SIZE

As you would expect, the RAM disk will not accept more information than it can hold and there is a limit of 12 files in the directory. The Copy Files command terminates abruptly, but benignly, if you try to copy a thirteenth file. This occurs even with blocks of free memory remaining. For a way around the directory size limit, use the ProDOS command to make subdirectories. The subject of subdirectories is well documented in the ProDOS.

SIDE EFFECTS

In general, /RAM has no side effects to disturb the rest of the system. Activating the 80-column display mode does not affect the operation of /RAM. The same kinds of files that would be stored on a disk can be placed in /RAM. The existence of files in /RAM will not disturb copy or formatting procedures for disks in other drives. Hi-Res pictures are unaffected by files in /RAM. However, double Hi-Res pictures use the same memory locations as /RAM files, so loading a double Hi-Res image with files in /RAM makes garbage of them both.

LUCK — A Lower to Uppercase Converter

Most older Apple II and II Plus computers do not have the capability to display lowercase characters. This easy-to-use utility converts the lowercase characters in an Applesoft program to uppercase so you can write a single program for both old and new machines.

by Kirk Paterson

Lowercase adapters and 80-column cards with lowercase are very useful. Without one of these handy devices, a listing that was written with lowercase characters is displayed in a most confusing manner. If you have not had occasion to see such a mess, consider yourself lucky. the lowercase characters appear primarily as punctuation marks and numbers. Result: pure gibberish!

WHY YOU NEED LUCK

If your setup has the capability to put lowercase characters on the screen, you may not see the need for LUCK (Lower to Uppercase Keyboard). However, I had recently written a routine that I wanted to pass on to a friend, but as I was about to copy it onto a disk for him, I realized that his system didn't have lowercase capability. It also occurred to me that if my Videx card failed, my programs that used lowercase would become useless. This may never happen to you, but if you are afraid your luck might run out, here is a little program that will give you a bit more luck.

WHAT LUCK DOES

LUCK (**Listing 55**) is a short machine language program that will convert lowercase characters to uppercase in all the DATA, PRINT and REM statements in your Applesoft programs. This includes string assignments such as:

```
LET A$ = "this is a test"
```

and

```
DATA this, is, a, test
```

as well as string literals such as:

```
DATA "this", "is", "a", "test"
```

USING THE PROGRAM

After you have LOADed an Applesoft program, BRUN LUCK. The conversion will be so fast, your disk drive may still be turning when it's done. The program can convert more than 3K of solid lowercase in that short a time.

ENTERING THE PROGRAM

Please refer to Appendix A for help in entering this program. If you key it in from the Monitor, save it to disk with the command:

BSAVE LUCK,A$300,L$81

HOW LUCK WORKS

The program is well-documented, but let's cover the major functions here. The pointer to the beginning of your Applesoft program is determined and preserved at the unused zero page locations $FE and $FF. This pointer is then constantly updated to point to the next character in the program.

READ.A.BYTE then puts the byte indicated by the pointer in the Accumulator and examines it for a $00 (the flag for the end of a line), a $B2 (the token for REM), a $83 (the token for DATA), or a $22 (the token for a quotation mark). If any of these is found, the program branches to the appropriate subroutine. If none is found, the pointer is updated and the next character is read.

Tokens are one-byte codes for the keywords that you use as commands in Applesoft. (For a complete listing, see p. 24 of the BASIC Programming Reference Manual, or Appendix H.4 of the *Applesoft BASIC Programmer's Reference Manual for the IIc*.)

CHK.FOR.END looks at the next two bytes for zeros, which would indicate the end of the program. If three consecutive zeros are found, the end of the Applesoft program has been reached, and the LUCK program ends. If not, the Y-Register is set back to zero and a branch is taken to END.OF.LINE.

At END.OF.LINE the current pointer is increased by four to skip the two line-link bytes and the two line-numbers bytes. The program then branches back to the search routine.

Program flow branches to FND.REM.DATA when a REM or a DATA token has been found. The current pointer is moved ahead one character at a time, and each character is checked for lowercase via a JSR to CHK.FOR.LC. When a $00 is found, indicating the end of the line, the program branches through the routine that checks for the end of the program to the main body.

FOUND.QUOTE works in much the same manner as FND.REM.DATA except that the program also looks for a second quotation mark ("), which indicates the end of a string literal. As it is possible to end a PRINT statement without a closing quotation mark, finding a zero (end of line) will also cause a break out of this routine.

The subroutine CHK.FOR.LC is used by the other routines to find and convert lowercase characters. The lowercase characters are represented by ASCII codes from $60-$7F. Their uppercase counterparts are $20 less, so it is only necessary to subtract $20 from each one and store it back where it wass. ZZLEN simply calculates the length of the program.

An interesting bug crept into the program when I was writing it, and it actually took longer to track down and squash the bug than to write the program. When line numbers such as 290 and 8704 were translated into hex, they contained a byte that was exactly $22. This is the token for a quotation mark and was read as such. The following bytes were then reduced by $20, which created a real mess. It may be true that the "problem is in the software," but it pays to remember that the software includes the interpretation and not just the code.

LISTING 55: LUCK

```
;
;                LUCK
; LOWER TO UPPER CASE KEYBOARD
;       BY KIRK PATERSON
;     COPYRIGHT (C)  1985
;     BY MICROSPARC, INC.
;     CONCORD, MA  01742
;
;
;       Converts lower case letters in PRINT and REM
;     statements of Applesoft programs to upper case.
;
```

```
                                 ORG $300
                        ;
                        ;             EQUATES
                        ;

                        PRGBEG     EQU $67,68       ;Beginning of Applesoft Prog.
                        CURR.PNTREQU $FE,FF         ;Pointer to current char.

                        ;
                        ZZBEG
0300   A6 67                       LDX PRGBEG        ;Initialize CURR.PNTR
0302   CA                          DEX               ;  less 1
0303   86 FE                       STX CURR.PNTR     ;every line of APPLESOFT.
0305   A6 68                       LDX PRGBEG+1
0307   86 FF                       STX CURR.PNTR+1

0309   A0 00                       LDY #0            ;Index.
030B   4C 3B 03                    JMP END.OF.LINE

                        ;
                        READ.A.BYTE
030E   B1 FE                       LDA (CURR.PNTR),Y ;Look at the next char.
0310   F0 15                       BEQ CHK.FOR.END   ;Zero? (End of line)
0312   C9 B2                       CMP #$B2          ;  Is it a REM?
0314   F0 33                       BEQ FND.REM.DATA  ; Yes, then convert it.
0316   C9 83                       CMP #$83          ;  Is it a DATA?
0318   F0 2F                       BEQ FND.REM.DATA  ; Yes, then convert it.
031A   C9 22                       CMP #$22          ;  Is it a quote?
031C   F0 3E                       BEQ FOUND.QUOTE   ;Then go to quote subrou.

                        NEXT.BYTE
031E   E6 FE                       INC CURR.PNTR  ;None of the above; carry o.
0320   D0 02                       BNE NB1          ;Watch for pointer reaching
0322   E6 FF                       INC CURR.PNTR+1  ;the end of a page (xxFF).

0324   4C 0E 03         NB1        JMP READ.A.BYTE

                        ;
                        CHK.FOR.END
0327   C8                          INY               ;Check for a second $00.
0328   B1 FE                       LDA (CURR.PNTR),Y   ; be a pointer so,
032A   F0 04                       BEQ CFE1            ; look for a third one.

032C   88                          DEY              ;If there was only one then
032D   4C 3B 03                    JMP END.OF.LINE   ;  not end of program

0330   C8               CFE1       INY               ; Is there a third zero?
0331   B1 FE                       LDA (CURR.PNTR),Y
0333   F0 05                       BEQ CFE2          ; End of program reached.

0335   88                          DEY               ;  Else restore index,
0336   88                          DEY
0337   4C 3B 03                    JMP END.OF.LINE   ;   and go to next line.

033A   60               CFE2       RTS               ;ALL DONE!!

                        ;
                        END.OF.LINE
```

```
033B  A5 FE             LDA CURR.PNTR      ;The link to the next line
033D  18                CLC               ;line number take up four bs
033E  69 04             ADC #4            ;so, increment the CURR.PNT4
0340  90 02             BCC EOL1            ;Crossing page boundary?
0342  E6 FF             INC CURR.PNTR+1     ;Bump the HI byte if so.
0344  85 FE    EOL1     STA CURR.PNTR       ;  Store the LO byte
0346  4C 1E 03          JMP NEXT.BYTE       ;and go to the next char.
               ;
               FND.REM.DATA
0349  E6 FE    FRD1     INC CURR.PNTR       ;Look at next byte.
034B  D0 02             BNE FRD2
034D  E6 FF             INC CURR.PNTR+1
034F  B1 FE    FRD2     LDA (CURR.PNTR),Y
0351  F0 06             BEQ FRD3            ;If $00, EOL reached.
0353  20 27 03          JSR CHK.FOR.LC      ;Else, check for LC char.
0356  4C 49 03          JMP FRD1            ;and go on to next char.
0359  4C 27 03 FRD3     JMP CHK.FOR.END   ;EOL--Check end of program
               ;
               FOUND.QUOTE
035C  E6 FE             INC CURR.PNTR     ;Bump CURR.PNTR to point
035E  D0 02             BNE FQ1           ;  at the next byte.
0360  E6 FF             INC CURR.PNTR+1
0362  B1 FE    FQ1      LDA (CURR.PNTR),Y ;Get the next character.
0364  F0 D5             BEQ END.OF.LINE     ;If=$00, closing quote o.
0366  C9 22             CMP #$22            ;Is it the closing quote?
0368  F0 06             BEQ FQ2             ;Yes; end subroutine.
036A  20 27 03          JSR CHK.FOR.LC      ;No; look for lowercase.
036D  4C 5C 03          JMP FOUND.QUOTE     ;and go to next charact.
0370  4C 1E 03 FQ2      JMP NEXT.BYTE       ;Quote closed; continue.
               ;
               CHECK.FOR.LC
0373  C9 60             CMP #$60            ;Below lower case ASCII?
0375  90 09             BCC CFL1            ;Yes; do nothing and re.
0377  C9 80             CMP #$80            ;Lower case ASCII?
0379  B0 05             BCS CFL1            ;Yes; do nothing and re.
037B  38                SEC                 ;Lower case so convert
037C  E9 20             SBC #$20            ;to the Upper Case ASCII
```

```
037E  91 FE                STA (CURR.PNTR),Y ;  and  stuff it back.

0380  60        CFL1       RTS


                ;
                ZZLEN      EQU ZZLEN-ZZBEG  ;Show length of program
```

```
000  ERRORS

0300  HEX START OF OBJECT
0380  HEX END OF OBJECT
0081  HEX LENGTH OF OBJECT
954B  HEX END OF SYMBOLS
```

Mini-Assembler Switch

Here's an easy way to use the Mini-Assembler that comes with Language Card Integer BASIC under DOS 3.3. There's no need for assembly language programs or EXEC files, and your Applesoft program and pointers are preserved.

by Charles Gilbert

In his article "Exec Mini-Assembler" (Nibble Vol. 4/No. 7), Bill Parker detailed an interesting method of using the Apple Mini-Assembler without destroying an Applesoft program or variable pointers. If you occasionally need to use the Mini-Assembler to write an assembly language routine for use with Applesoft, Exec Mini-Assembler offers a way to switch back and forth between Applesoft (ROM) and the Mini-Assembler (RAM card/Integer BASIC and old Monitor).

Unfortunately, Parker's method involves two EXEC files and an assembly language program on disk, not to mention yet another file (Applesoft Pointers) that is created when you use the EXEC files. Another problem with this method is that it uses an assembly language routine. There is only one small area of memory ($300-$3CF) that is completely safe to use for an assembly subroutine, because it will never be overwritten by Applesoft and/or DOS. If you use the Exec Mini-Assembler, then Parker's program is located in this area, so you have to use an unsafe area of memory for assembly language routines. Alternatively, you may relocate Bill's machine code in the unsafe area and use page 3 for your subroutine, but then you have to change the EXEC file to call the new location.

The Mini-Assembler switch requires no disk access at all. With Integer BASIC loaded into the RAM card area of memory, take the following steps to go from Applesoft to the Mini-Assembler

1. Exit Applesoft via CALL -151.
2. Type C080 followed by a <RETURN>. This turns the RAM card memory on, but does not initialize Integer BASIC.
3. Enter the Mini-Assembler with F666G.

At this point you may use the Mini-Assembler as usual. You may use one of two methods to return to Applesoft. The easier way is to use <RESET>. Since a warm <RESET> returns you to the last active BASIC, and Integer BASIC was never initialized, you will find yourself back in Applesoft with your program and all variables intact.

<RESET> may not work with certain hardware configurations, however. If <RESET> doesn't work for your system (or if you simply dislike using it), use the following method:

1. Exit the Mini-Assembler with $FF69G.
2. Type C081 followed by a <RETURN>. This turns off the RAM card area of memory and turns on the Applesoft ROMs.
3. Use <CTRL>C to reenter Applesoft.

For those who are interested in why as well as how this works, the important point is that the memory on the RAM card is enabled and disabled without using the FP or INT commands. This is accomplished by directly accessing the soft switches that control the status of upper memory access. When you type C080 or C081 followed by a <RETURN>, the Monitor (old or new) accesses that memory location and prints the contents of the location to the screen. In this case, the value that is returned to the screen is of no importance. Accessing the location sets the soft switch associated with the location.

Text Ups and Downs

Combine Applesoft and machine language to let you take a quick look at your DOS 3.3 text files without booting up your word processor. You can scroll the text up or down without splitting words at the ends of lines. If you have a printer, you can dump the screen with the touch of a key.

by Chester H. Page

TEXT.VIEWER (**Listing 57**) is an Applesoft program for loading a sequential text file and viewing it quickly. It uses SCROLL (**Listing 56**), a high-speed, machine language routine that allows two-way scrolling and prevents broken words at line ends.

USING TEXT VIEWER

When TEXT.VIEWER is run, it loads in SCROLL and requests the name of the file to be loaded. If you enter a question mark (?) for the file name, you will get the disk catalog. It then loads the file and displays the first 24 lines. The right and left arrow keys let you scroll up and down, respectively, one line at a time. The semicolon (;) and slash (/) do 12-line (half-screen) scrolling. Scrolling past the end of the text in either direction is automatically prevented. At any time, the text on display can be dumped to the printer by pressing P. Pressing N returns you to the file name prompt so that a new file can be loaded.

ENTERING THE PROGRAMS

Please refer to Appendix A for help in entering **Listing 56**. If you key it in from the Monitor, save it to disk with the command:

BSAVE SCROLL,A$1000,L$1BD

To key in the Applesoft driver program TEXT.VIEWER type in **Listing 57** as shown and save it to disk with the command:

SAVE TEXT.VIEWER

HOW THEY WORK

The machine language program, SCROLL, provides the power for TEXT.VIEWER. After the text file is loaded, SCROLL searches for potential line ends. At each of these, backward steps are taken until a space character is found. The space character is replaced with a <RETURN> and the line search resumes.

At any stage of the display, there must be a "next top line" and a "next bottom line" waiting for the next scroll command. These lines start immediately after a <RETURN> character and are located by pointers. Since scrolling up brings the previous next bottom line into the window, the first step in scrolling up is to advance the next bottom line pointer before scrolling. If the next bottom line would be past the end of the text, an RTS halts the scrolling. After scrolling, a new top line pointer is set up.

Scrolling down uses the reverse procedure. The next top line pointer is corrected before scrolling, and the next bottom line pointer is adjusted after scrolling. If the next top line would be ahead of the beginning of the text, an RTS interrupts the scrolling operation. These safety stops make use of a <RETURN> inserted at the beginning of the text and a zero at the end.

Listing 56: SCROLL

SOURCE FILE -

```
                           ;
                           ;        SCROLL
                           ;     BY CHET PAGE
                           ; COPYRIGHT (C) 1985
                           ; MICROSPARC, INC
                           ; CONCORD, MA  01742
                           ;
                           ;
                                    ORG $1000
                           STORE    EQU $4A
                           BOTTOM   EQU $D7
                           RDKEY    EQU $FD0C
                           VTABZ    EQU $FC24
                           BASCALC  EQU $FBC1
                           COUT     EQU $FDED
                           WNDTOP   EQU $22
                           WNDBTM   EQU $23
                           WNDWID   EQU $21
                           BASL     EQU $28
                           BASH     EQU $29
                           BAS2L    EQU $2A
                           BAS2H    EQU $2B
                           SCRNTP   EQU $6
                           SCRNBM   EQU $8
                           ASAV     EQU $CE
1000  A0 00                         LDY #0
1002  84 4A                         STY STORE
1004  A9 12                         LDA #$12
1006  85 D7                         STA BOTTOM
1008  85 4B                         STA STORE+1
100A  60                            RTS
100B  A0 00                         LDY #0
100D  A9 8D                         LDA #$8D
100F  91 4A                         STA (STORE),Y
1011  E6 4A                         INC STORE
1013  20 0C FD             LOAD     JSR RDKEY
1016  09 80                         ORA #$80
1018  C9 A0                         CMP #$A0
101A  B0 04                         BCS ]1
101C  C9 8D                         CMP #$8D
101E  D0 F3                         BNE LOAD
1020  91 4A                ]1       STA (STORE),Y
1022  E6 4A                         INC STORE
1024  D0 ED                         BNE LOAD
1026  E6 4B                         INC STORE+1
1028  4C 13 10                      JMP LOAD
102B  A9 8D                SCALL    LDA #$8D           ;SECOND CALL
102D  A0 01                         LDY #1
102F  91 4A                         STA (STORE),Y
1031  88                            DEY
1032  A9 00                         LDA #0
1034  91 4A                         STA (STORE),Y
1036  85 4A                         STA STORE
1038  A5 D7                         LDA BOTTOM
```

```
103A  85 4B              STA STORE+1
103C  A0 00     EDIT     LDY #0
103E  B1 4A     EDIT1    LDA (STORE),Y
1040  C9 00              CMP #0
1042  F0 22              BEQ DONE
1044  C9 8D              CMP #$8D
1046  F0 0C              BEQ ENDLN
1048  C8                 INY
1049  C4 21              CPY WNDWID
104B  90 F1              BCC EDIT1
104D  88        BACK     DEY
104E  B1 4A              LDA (STORE),Y
1050  C9 A0              CMP #$A0
1052  D0 F9              BNE BACK
1054  A9 8D     ENDLN    LDA #$8D
1056  91 4A              STA (STORE),Y
1058  C8                 INY
1059  18                 CLC
105A  98                 TYA
105B  65 4A              ADC STORE
105D  85 4A              STA STORE
105F  90 DB              BCC EDIT
1061  E6 4B              INC STORE+1
1063  4C 3C 10           JMP EDIT
1066  85 08     DONE     STA SCRNBM
1068  A5 D7              LDA BOTTOM
106A  85 09              STA SCRNBM+1
106C  60                 RTS              ;READY TO GO
106D  20 49 11  UP       JSR NEWBTM
1070  A5 22              LDA WNDTOP
1072  85 CE              STA ASAV
1074  20 24 FC           JSR VTABZ
1077  20 8A 10           JSR NXTLN
107A  A4 21     INIT     LDY WNDWID
107C  88                 DEY
107D  B1 28     NXTCHR   LDA (BASL),Y
107F  91 2A              STA (BAS2L),Y
1081  88                 DEY
1082  10 F9              BPL NXTCHR
1084  20 8A 10           JSR NXTLN
1087  4C 7A 10           JMP INIT
108A  A5 28     NXTLN    LDA BASL
108C  85 2A              STA BAS2L
108E  A5 29              LDA BASH
1090  85 2B              STA BAS2H
1092  A5 CE              LDA ASAV
1094  18                 CLC
1095  69 01              ADC#1
1097  C5 23              CMP WNDBTM
1099  B0 06              BCS LDBTM
109B  85 CE              STA ASAV
109D  20 24 FC           JSR VTABZ
10A0  60                 RTS
10A1  68        LDBTM    PLA
10A2  68                 PLA
10A3  A0 FF              LDY #$FF
10A5  C8        LB       INY
10A6  B1 08              LDA (SCRNBM),Y
```

```
10A8  91 28               STA (BASL),Y
10AA  C9 8D               CMP #$8D
10AC  D0 F7               BNE LB
10AE  84 CE               STY ASAV
10B0  E6 CE               INC ASAV
10B2  A9 A0               LDA #$A0
10B4  91 28      ]1       STA (BASL),Y
10B6  C8                  INY
10B7  C4 21               CPY WNDWID
10B9  90 60               BCC ]1
10BB  A0 FF      NEWTOP   LDY #$FF
10BD  C8         NT       INY
10BE  B1 06               LDA (SCRNTP),Y
10C0  C9 8D               CMP #$8D
10C2  D0 F9               BNE NT
10C4  C8                  INY
10C5  18                  CLC
10C6  98                  TYA
10C7  65 06               ADC SCRNTP
10C9  85 06               STA SCRNTP
10CB  90 02               BCC RTS1
10CD  E6 07               INC SCRNTP+1
10CF  60         RTS1     RTS
10D0  20 65 11   DOWN     JSR NEWTP2
10D3  A5 23               LDA WNDBTM
10D5  38                  SEC
10D6  E9 01               SBC #1
10D8  85 CE               STA ASAV
10DA  20 24 FC            JSR VTABZ
10DD  20 F0 10            JSR NXTLN2
10E0  A4 21      INIT2    LDY WNDWID
10E2  88                  DEY
10E3  B1 28      NXTCR2   LDA (BASL),Y
10E5  91 2A               STA (BAS2L),Y
10E7  88                  DEY
10E8  10 F9               BPL NXTCR2
10EA  20 F0 10            JSR NXTLN2
10ED  4C E0 10            JMP INIT2
10F0  A5 28      NXTLN2   LDA BASL
10F2  85 2A               STA BAS2L
10F4  A5 29               LDA BASH
10F6  85 2B               STA BAS2H
10F8  A5 CE               LDA ASAV
10FA  38                  SEC
10FB  E9 01               SBC #1
10FD  C5 22               CMP WNDTOP
10FF  30 07               BMI LDTOP
1101  85 CE               STA ASAV
1103  20 24 FC            JSR VTABZ
1106  38                  SEC
1107  60                  RTS
1108  68         LDTOP    PLA
1109  68                  PLA
110A  A0 FF               LDY #$FF
110C  C8         LT       INY
110D  B1 06               LDA (SCRNTP),Y
110F  91 28               STA (BASL),Y
1111  C9 8D               CMP #$8D
```

```
1113  D0 F7              BNE LT
1115  84 CE              STY ASAV
1117  E6 CE              INC ASAV
1119  A9 A0              LDA #$A0
111B  91 28     ]1       STA (BASL),Y
111D  C8                 INY
111E  C4 21              CPY WNDWID
1120  90 F9              BCC [1
1122  A0 00              LDY #0
1124  C6 08              DEC SCRNBM
1126  A5 08              LDA SCRNBM
1128  C9 FF              CMP #$FF
112A  D0 02              BNE NB
112C  C6 09              DEC SCRNBM+1
112E  C6 08     NB       DEC SCRNBM
1130  A5 08              LDA SCRNBM
1132  C9 FF              CMP #$FF
1134  D0 02              BNE ]1
1136  C6 09              DEC SCRNBM+1
1138  B1 08     ]1       LDA (SCRNBM),Y
113A  C9 8D              CMP #$8D
113C  D0 F0              BNE NB
113E  E6 08              INC SCRNBM
1140  A5 08              LDA SCRNBM
1142  C9 00              CMP #$0
1144  D0 89              BNE RTS1
1146  E6 09              INC SCRNBM+1
1148  60                 RTS
1149  A0 FF     NEWBTM   LDY #$FF
114B  C8        ]1       INY
114C  B1 08              LDA (SCRNBM),Y
114E  C9 00              CMP #0
1150  D0 03              BNE ]2
1152  68                 PLA
1153  68                 PLA
1154  60                 RTS
1155  C9 8D     ]2       CMP #$8D
1157  D0 F2              BNE [1
1159  C8                 INY
115A  18                 CLC
115B  98                 TYA
115C  65 08              ADC SCRNBM
115E  85 08              STA SCRNBM
1160  90 02              BCC RTS2
1162  E6 09              INC SCRNBM+1
1164  60        RTS2     RTS
1165  A0 00     NEWTP2   LDY #0
1167  A5 06              LDA SCRNTP
1169  C9 01              CMP #1
116B  D0 DE              BNE [1
116D  A5 07              LDA SCRNTP+1
116F  C5 D7              CMP BOTTOM
1171  D0 03              BNE ]1
1173  68                 PLA
1174  68                 PLA
1175  60                 RTS
1176  C6 06     ]1       DEC SCRNTP
1178  A5 06              LDA SCRNTP
```

```
117A  C9 FF              CMP #$FF
117C  D0 02              BNE NT2
117E  C6 07              DEC SCRNTP+1
1180  C6 06     NT2      DEC SCRNTP
1182  A5 06              LDA SCRNTP
1184  C9 FF              CMP #$FF
1186  D0 02              BNE ]2
1188  C6 07              DEC SCRNTP+1
118A  B1 06     ]2       LDA (SCRNTP),Y
118C  C9 8D              CMP #$8D
118E  D0 F0              BNE NT2
1190  E6 06              INC SCRNTP
1192  A5 06              LDA SCRNTP
1194  C9 00              CMP #0
1196  D0 CC              BNE RTS2
1198  E6 07              INC SCRNTP+1
119A  60                 RTS
119B  A9 FF     UMP      LDA #$FF
119D  85 CE              STA ASAV
119F  E6 CE     START    INC ASAV
11A1  A5 CE              LDA ASAV
11A3  C9 18              CMP #$18
11A5  F0 15              BEQ FIN
11A7  20 C1 FB           JSR BASCALC
11AA  A0 00              LDY #0
11AC  B1 28     ]1       LDA (BASL),Y
11AE  20 ED FD           JSR COUT
11B1  C8                 INY
11B2  C0 28              CPY #$28
11B4  90 F6              BCC [1
11B6  20 8E FD           JSR $FD8E
11B9  4C 9F 11           JMP START
11BC  60        FIN      RTS

000  ERRORS

1000  HEX START OF OBJECT
11BC  HEX END OF OBJECT
01BD  HEX LENGTH OF OBJECT
9475  HEX END OF SYMBOLS
```

Listing 57: TEXT.VIEWER

```
10  REM  **********************
20  REM  *     TEXT.VIEWER    *
30  REM  *     BY CHET PAGE   *
40  REM  * COPYRIGHT (C) 1985 *
50  REM  *  MICROSPARC, INC   *
60  REM  * CONCORD, MA 01742  *
70  REM  **********************
80 D$ =  CHR$ (4)
90  IF  PEEK (4096) <  > 160 THEN 120
100  IF  PEEK (4098) <  > 132 THEN 120
110  IF  PEEK (4099) = 74 THEN 130
120  PRINT D$"BLOAD SCROLL"
130  CALL 4096
```

```
140  HOME : VTAB 3: HTAB 14: PRINT "TEXT VIEWER": HTAB 14: PRINT
     "BY CHET PAGE": PRINT "* COPYRIGHT 1985 BY MICROSPARC, INC.
     *": VTAB 10: PRINT "ENTER FILE NAME (? FOR CATALOG)"
150  INPUT "";F$
160  IF F$ = "?" THEN  PRINT D$"CATALOG": GET A$: PRINT : GOTO
     140
170  ONERR  GOTO 400
180  PRINT D$"OPEN"F$
190  PRINT D$"READ"F$
200  INPUT K$
210  PRINT D$"CLOSE"F$: ONERR  GOTO 250
220  PRINT D$"OPEN"F$
230  PRINT D$"READ"F$
240  CALL 4107
250  PRINT D$"CLOSE"F$
260  CALL 4139
270  POKE 6,1: POKE 7, PEEK (215)
280  FOR I = 1 TO 24: CALL 4205: NEXT
290  POKE 6,1: POKE 7, PEEK (215)
300 KEY =  PEEK ( - 16384): POKE  - 16368,0
310  IF KEY = 149 THEN  CALL 4205
320  IF KEY = 136 THEN  CALL 4304
330  IF KEY = 187 THEN  FOR I = 1 TO 12: CALL 4205: NEXT
340  IF KEY = 175 THEN  FOR I = 1 TO 12: CALL 4304: NEXT
350  IF KEY = 206 THEN 130
360  IF KEY = 208 THEN  GOSUB 390
370  IF KEY <  > 155 THEN 300: REM  ESC TO QUIT
380  HOME : END
390  PRINT D$"PR#1": PRINT  CHR$ (9)"80N": CALL 4507: PRINT
     D$"PR#0": RETURN
400  PRINT D$"CLOSE": IF  PEEK (222) = 5 THEN  PRINT "NO SUCH
     FILE": PRINT "PRESS ANY KEY TO TRY AGAIN": GET K$: PRINT :
     GOTO 130
410  PRINT "ERROR #" PEEK (222)" IN LINE " PEEK (218) +  PEEK
     (219) * 256: END
```

Applewriter IIc

Use this utility to make a version of the popular word processor Applewriter // that will function properly on the enhanced IIe, IIc and IIGS. Inverse characters are displayed correctly, and 40-column mode can be manually selected for TV display. The program requires an Apple IIe or IIc operating under DOS 3.3, and Applewriter //.

by Steven Meuse

Applewriter IIc is the result of a whim. One lazy day it occurred to me that it would be fun to lie on the couch and process words in 40 columns on the family TV. Had I known then that I would spend a weekend modifying Applewriter // (the DOS 3.3 version designed for the IIe) to work on an Apple IIc, I might not have done it.

Foresight being what it is, the project progressed "just another half hour" at a time. There are no great mysteries to how it works (lots of disassembly), and a detailed description is beyond the scope of this article. I would like to share the program, though, and explain how to use it and what to expect from the patched version of Applewriter //.

PROBLEMS WITH APPLEWRITER //

Applewriter // (DOS 3.3 version) has a few problems with the Apple IIc because it's author bypassed the normal text output routines in the Apple Monitor and stored the characters directly into the video display buffer. While this creates faster displays, the Mousetext ROM in the Apple IIc takes exception to certain values being put in the display buffer, and it displays Mousetext instead of the intended (inverse) characters. (Mousetext is an addition to the Apple character set that contains icons for use in Lisa/Macintosh-like applications that use the mouse. It occupies space formerly used by a duplicate set of inverse, uppercase characters.)

One problem showed up on the status line, the bar at the top of the screen that shows in inverse characters the amount of available memory, the length of the current document, the file name and other important information. Another problem showed up when the cursor was over a capital letter; this was also displayed as Mousetext.

Also, a feature was lacking. There is an 80/40 switch on the Apple IIc that informs the software whether to use the 80- or 40-column screen. Our TV isn't quite up to snuff when it comes to displaying 80 columns, so this was an important feature to include. Fortunately, Applewriter // can display 40 columns, but the display decision is made automatically, based on whether there is an 80-column card. I had to fool Applewriter into using its 40-column screen routines even when there is 80-column firmware, as there is in the IIc. The result is AWCONVERT (**Listing 58**), an Applesoft program that will automatically load the necessary Applewriter // files, modify them to provide normal text and 40-column capability, and then save them on disk.

USING AWCONVERT

To use AWCONVERT, just run it and follow the prompts. Insert a backup of your Applewriter // disk and press <RETURN>. The rest is automatic and takes a few moments to complete. The program tells you when the conversion is complete.

A few words of caution are in order. Use the program only on a backup of your Applewriter // disk and double-check your typing before running the program. If disaster should strike, and the modified Applewriter does not work, use FID or a similar file transfer program to copy two files from your master to your backup disk, and everything will be back to normal. Those two files (and the only two that AWCONVERT modifies) are OBJ.BOOT and OBJ.APWRT][F.

NEW FEATURES

What are the new features of Applewriter IIc? Of course, the converted version works on the IIe the same as it always did. Now it works on the enhanced IIe, IIc and IIGS the way it does on the IIe.

In addition, the modified Applewriter disk will now recognize when the 80/40 column switch is pressed in during bootup, and it will use the 40-column display. The 40-column display can also be accessed on the IIe by pressing 4 during bootup. If Applewriter is run on a 128K Apple (IIc, IIGS or IIe with extended 80-column card), it will recognize and use the extra memory, regardless of the 80/40 column choice.

The one difference between the normal and patched Applewriter // is the first message you get on bootup. Instead of:

```
(For help while editing, press open-Apple and "?")
Press RETURN:
```

it now says:

```
Apple //c version
```

The memory saved by shortening the message was used for the patches. In addition, this gives you a sure-fire way of knowing which version you are using. Just remember to press <RETURN> to go on, and remember that open-Apple and ? gives you the help menu. Now that the work is done, it is rather pleasant to sit here on the couch and use Applewriter on the Apple IIc. Somehow these soft cushions make it all worthwhile.

ENTERING THE PROGRAM

To key in AWCONVERT, type in **Listing 58** as shown and save it to disk with the command:

SAVE AWCONVERT

LISTING 58: AWCONVERT

```
10  REM  **********************
20  REM  *      AWCONVERT     *
30  REM  *  BY STEVEN MEUSE   *
40  REM  * COPYRIGHT (C) 1985 *
50  REM  * MICROSPARC, INC    *
60  REM  * CONCORD, MA 01742  *
70  REM  **********************
80  REM CONVERT APPLEWRITER //
90  REM FOR //C COMPATIBILITY
100  HIMEM: 6400:D$ =  CHR$ (4): TEXT : HOME : VTAB 9
110  PRINT "Insert a COPY of your Applewriter disk"
120  PRINT : PRINT "and press [RETURN]. ";: GET A$: PRINT
130  PRINT D$"BLOAD OBJ.BOOT"
140  POKE 7383,194: POKE 7384,30
150  FOR X = 7874 TO 7904: READ L: POKE X,L: NEXT
160  DATA 32,234,29,173,0,192,201,180,208,5,141,16,
      192,240,10,173,192,251,208,10
170  DATA  44,96,192,16,5,169,0,141,62,29,96
180  PRINT D$"UNLOCK OBJ.BOOT"
```

```
190  PRINT D$"BSAVE OBJ.BOOT,A$1C00,L$2E1"
200  PRINT D$"LOCK OBJ.BOOT"
210  PRINT D$"BLOAD OBJ.APWRT][F"
220  POKE 7129,76: POKE 7130,88: POKE 7131,80
230  POKE 7266,97: POKE 7267,80
240  POKE 12497,99: POKE 12498,80
250  FOR X = 17989 TO 18031: READ L: POKE X,L: NEXT
260  DATA
     193,240,240,236,229,160,175,175,227,160,246,229,242,243,
     233,239,238,160,0,201
270  DATA
     96,176,2,41,63,145,40,96,164,36,72,10,10,48,4,104,41,191,
     72,104
280  DATA 76,222,37
290  PRINT D$"UNLOCK OBJ.APWRT][F"
300  PRINT D$"BSAVE OBJ.APWRT][F,A$1900,L$30D1"
310  PRINT D$"LOCK OBJ.APWRT][F"
320  VTAB 20: PRINT "Conversion complete."
```

Beep Customizer

Use this Applesoft program under DOS 3.3 to produce a customized machine language tone routine. Every time a <CTR>G character is encountered, the custom tone will be sounded, instead of the usual bell sound.

by John Baumbach

Anyone who uses the Apple often has encountered the SYNTAX ERROR. At first, the accompanying beep is only mildly annoying, but after hearing it over and over, it can get downright irritating. I decided to change the Apple beep and allow easy modification of the new beep with Beep Customizer.

USING THE PROGRAM

When you run BEEP.CUSTOMIZER (**Listing 59**) the current settings for the beep will be displayed, along with a four-item menu as shown in **Figure 7**. If an A is entered, the program will ask you for the length and tone you wish for the beep. The program will only accept values from 1-255. Press <RETURN> if you want to use the current value.

FIGURE 7: *Menu Display*

```
*******************************************
*  BEEP  CUSTOMIZER   BY  JOHN  BAUMBACH  *
*  COPYRIGHT  1985   BY  MICROSPARC   INC *
*******************************************

CURRENT LENGTH:  64
CURRENT TONE   :  16


(A) MODIFY CURRENT BEEP
(B) HEAR CURRENT BEEP
(C) SAVE CURRENT BEEP/QUIT
(D) QUIT

ENTER =>
```

To hear the beep that is currently defined by the length and tone shown on the screen, press B. If you don't hear anything, check the DATA statement in **line 670**, or raise the tone a bit. The smaller the tone value, the higher the pitch.

If you press C, you will be asked for the location to place the beep routine in memory. If you are not sure, you can accept the default value (768) by pressing <RETURN>. The machine language program is then saved onto your disk and it is also put into memory. Entering D will do the same thing, but the machine language program is not saved to your disk. To exit the program without doing anything, press <ESC>.

Now pressing <CTRL>G will output your new beep. You will also notice that getting a SYNTAX ERROR is more pleasurable.

Once you have saved the machine language code for your new beep to disk, you can just type BRUN APPLE.BEEP to put the beep into memory. Then you're ready to go! A

<RESET> will disable the custom beep, but you can restore it by doing a CALL to your starting address.

ENTERING THE PROGRAM

To key in the program, enter the Applesoft code shown in **Listing 59** and save it to disk with the command:

SAVE BEEP.CUSTOMIZER

There is no need to enter the code shown in **Listing 60** since BEEP.CUSTOMIZER will automatically generate the machine language code and save it for you under the name APPLE.BEEP.

HOW IT WORKS

Line 80 in **Listing 59** is a DIM statement, that DIMensions array A for the assembly language program. The program then sets up the title page in **lines 100-140** and loads the data for the assembly language program into array A (**line 180**), which was previously defined in **line 80. Lines 190-200** display the current length and tone. The variable A(19) contains the length (which starts as 64), and the tone (which starts as 16) is contained in the variable A(21). The numbers 64 and 16 were arbitrarily chosen to define the beep.

Lines 210-240 display the menu and ask you for a selection. More about what each selection does later. The part of the program that modifies the beep (**lines 330-410**) asks you for a new length and new tone. When the new length is entered, it replaces the old length in the variable A(19), and the new tone replaces the old tone in the variable A(21).

The next part of the program (**lines 440-470**) outputs the beep. The separate machine language program (**Listing 60**) outputs the beep itself, and since there is no permanent place for it right now, for the moment, it is put it into a "safe" place at decimal 750-767. This location is safe for the time being, but it will be overwritten by other things soon. The machine language program itself was read into the variable A previously (**line 180**), and is POKED into memory now.

Line 490 saves the assembly language program to disk under the name APPLE.BEEP. It first calls a locator routine in **line 560**, saves APPLE.BEEP, and then exits to **line 530. Lines 520-530** also call the locator routine, and then quit the program, after activating the machine language routine.

In the locator routine things get a little complicated, but bear with me. In **line 560**, the routine first gets the starting location in memory for the machine language beep routine. I usually put short machine language routines at location 768 since it causes very few problems, so this is the default value (**line 570**). The selected location is then output (**line 590**). The formula in **lines 600-610** alters the machine language program a bit by adjusting pointers. In **lines 620-650**, the entire machine language program is POKEd into memory at the final location and **line 660** passes program control from the locator routine back to the main calling routine. Finally, the machine language program in its Applesoft BASIC form is seen in the DATA statement (**line 670**).

Listing 59 shows a sample APPLE.BEEP program generated by BEEP.CUSTOMIZER. This version specifies a starting address of 768, tone of 16, and length of 64. The assembly format is provided to help you better understand how it works. The actual beep program starts at $30B; the code at $300 changes the DOS 3.3 output hook (CSWL,CSWH) so that it points to the beep routine.

Thereafter, before each character is output it is checked by the CMP #$87 statement at $30B. If the character in the A-Register is a <CTRL>G, the rest of the beep routine is executed. Otherwise, a branch to $324 is made, and the character is output normally.

The BEEP routine uses two built-in Applesoft routines to produce a tone. One produces a delay based on the contents of the Accumulator and the other produces a click from the speaker. The contents of Y, or the value you specified for tone length in

BEEP.CUSTOMIZER, determine how many speaker clicks to produce, while the contents of A, provided as the tone frequency in BEEP.CUSTOMIZER, determine the amount of delay between clicks.

LISTING 59: BEEP.CUSTOMIZER

```
10   REM **********************
20   REM *  BEEP.CUSTOMIZER   *
30   REM *  BY JOHN BAUMBACH  *
40   REM * COPYRIGHT (C) 1985 *
50   REM * BY MICROSPARC, INC *
60   REM * CONCORD, MA  01742 *
70   REM **********************
80   DIM A(36)
90   TEXT : HOME
100  FOR L = 1 TO 36:A$ = A$ + "*": NEXT L
110  HTAB 3: PRINT A$
120  HTAB 3: PRINT "* BEEP CUSTOMIZER BY JOHN BAUMBACH *"
130  HTAB 3: PRINT "* COPYRIGHT 1985 BY MICROSPARC INC *"
140  HTAB 3: PRINT A$
150  PRINT
160  POKE 34,7
170  REM  READ A.L. INTO "A"
180  FOR L = 1 TO 36: READ A(L): NEXT
190  VTAB 6: HTAB 1: CALL  - 958: PRINT "CURRENT LENGTH: ";:
     INVERSE : PRINT A(19): NORMAL
200  PRINT "CURRENT TONE  : ";: INVERSE : PRINT A(21): NORMAL
210  PRINT : PRINT : PRINT : PRINT "(A) MODIFY CURRENT BEEP"
220  PRINT "(B) HEAR CURRENT BEEP"
230  PRINT "(C) SAVE CURRENT BEEP/QUIT"
240  PRINT "(D) QUIT"
250  PRINT : PRINT "ENTER => ";: GET R$
260  IF R$ = "A" THEN 330
270  IF R$ = "B" THEN 440
280  IF R$ = "D" THEN 520
290  IF R$ = "C" THEN 490
300  IF R$ =  CHR$ (27) THEN  PRINT  CHR$ (92): PRINT : GOTO 680
310  GOTO 190
320  REM MODIFY BEEP
330  PRINT : PRINT : PRINT "<RETURN> TO ACCEPT CURRENT VALUE;
     ENTER (0-255)": PRINT
340  INPUT "ENTER NEW LENGTH => ";LNGTH$: IF LNGTH$ = "" THEN
     LNGTH$ =  STR$ (A(19))
350  IF  VAL (LNGTH$) <  = 0 THEN 340
360  IF  VAL (LNGTH$) > 255 THEN 340
370  INPUT "ENTER NEW TONE => ";TNE$: IF TNE$ = "" THEN TNE$ =
     STR$ (A(21))
380  IF  VAL (TNE$) <  = 0 THEN 370
390  IF  VAL (TNE$) > 255 THEN 370
400 A(19) =  VAL (LNGTH$)
410 A(21) =  VAL (TNE$)
420  GOTO 190
```

```
430  REM OUTPUT BEEP
440  PRINT : PRINT "LISTEN....   "
450  FOR L = 1 TO 100: NEXT
460  REM "CALL 750" TEMPORARY BEEP
470  FOR L = 750 TO 767: POKE L,A(L - 734): NEXT : CALL 750: FOR
     L = 1 TO 300: NEXT : GOTO 190
480  REM SAVE BEEP
490  HOME : PRINT : PRINT "SAVE CURRENT BEEP:": PRINT : GOSUB
     560: PRINT : PRINT  CHR$ (4);"BSAVE
     APPLE.BEEP,A";LCT;",L37"
500  PRINT : PRINT "BEEP SAVED ON DISK": PRINT : GOTO 530
510  GOTO 190
520  HOME : PRINT "QUIT PROGRAM:": PRINT : GOSUB 560
530  PRINT : PRINT "BEEP INSTALLED.": CALL LCT: GOTO 680
540  REM PUT BEEP IN MEM AT
550  REM   LOCATION "LCT"
560  PRINT : INPUT "ENTER A LOCATION TO PUT THE BEEP
     MODIFIER ROUTINE  (DEFAULT=768, $300)   ---=> ";LCT$
570  IF LCT$ = "" THEN LCT = 768: GOTO 590
580 LCT =  VAL (LCT$)
590  PRINT : PRINT "LOCATION IS ";LCT: PRINT
600 LCT = LCT + 11:B = LCT -  INT (LCT / 256) * 256:HI =  INT
     (LCT / 256):A(2) = B:A(6) = HI
610 LCT = LCT - 11
620 X = 0
630  FOR L = LCT TO LCT + 35
640 X = X + 1
650  POKE L,A(X): NEXT
660  RETURN
670  DATA 169,11,133,54,169,3,133,55,76,234,3,201,135,208,
     18,152,72,160,64,169,16,32,168,252,173,48,192,136,208,245,1
     04,168,96,76,240,253
680  TEXT : END
```

LISTING 60: APPLE.BEEP

```
                       ;
                       ;**********************
                       ;*     APPLE.BEEP     *
                       ;*  BY JOHN BAUMBACH  *
                       ;* COPYRIGHT (C) 1985 *
                       ;* BY MICROSPARC, INC *
                       ;* CONCORD, MA  01742 *
                       ;**********************
                                ORG $300
                       CSWL     EQU $36
                       CSWH     EQU $37
                       RECON    EQU $3EA
                       CTRLG    EQU $87
                       SPKR     EQU $C030
                       WAIT     EQU $FCA8
                       COUT1    EQU $FDF0
                       ;
    0300   A9 0B       BPINIT   LDA #BEEP         ;REDIRECT CHROUT VECTOR
```

```
0302  85 36                 STA CSWL      ;TO GO THROUGH BEEP
0304  A9 03                 LDA #BEEP/
0306  85 37                 STA CSWH
                   ;
0308  4C EA 03              JMP RECON     ;RECONNECT DOS VECTORS
                   ;
030B  C9 87        BEEP     CMP #CTRLG    ;CHROUT DIRECTED HERE
030D  D0 12                 BNE NORM      ;CHAR OUTPUT IF NOT CTRL-G
030F  98                    TYA           ;SAVE Y-REG. ON STACK
0310  48                    PHA
0311  A0 40                 LDY #$40      ;TONE LENGTH FR. CUSTOMIZER
0313  A9 10        BPLOOP   LDA #$10      ;TONE FREQ. FR. CUSTOMIZER
0315  20 A8 FC              JSR WAIT
0318  AD 30 C0              LDA SPKR      ;CLICK SPEAKER
031B  88                    DEY
031C  D0 F5                 BNE BPLOOP
031E  68                    PLA           ;RESTORE Y-REG.
031F  A8                    TAY
0320  60                    RTS
0321  4C F0 FD     NORM     JMP COUT1     ;OUTPUT CHAR
                   ;

000  ERRORS

0300  HEX START OF OBJECT
0323  HEX END OF OBJECT
0024  HEX LENGTH OF OBJECT
95AB  HEX END OF SYMBOLS
```

Status Seeker

Status Seeker gives you the status of 11 functions. Put it in your programs to provide you with valuable information.

by Paul Raymer

As most folks who program do, I have been collecting PEEKs, POKEs and CALLs ever since I figured out what they do. I have them in a big notebook and add to them when I find new ones. It was just a matter of digging some out for this program.

WHAT DOES STATUS SEEKER DO?

When STATUS.SEEKER runs, your computer determines the status of 11 functions for you. As it is, it makes an interesting demonstration. When placed at the end of a program or when parts of it are used in your program, it could provide valuable information.

ENTERING THE PROGRAM

This is one of those "have faith" listings. Just type in the Applesoft program as shown in **Listing 61** (watching the parentheses closely) and save it to disk with the command:

SAVE STATUS.SEEKER

Then watch the program do its thing by typing RUN STATUS.SEEKER

EXPERIMENT!

Assign a value to AZ$ or X% in a new **line 145**. Change the speed to any number from 1-255 and watch the program change. Add extra REM statements, and watch the program length change. Move your joystick or paddles or use a Koala Pad to watch the paddle value numbers change.

LISTING 61: STATUS.SEEKER

```
10   REM **********************
20   REM *    STATUS.SEEKER    *
30   REM *    BY PAUL RAYMER   *
40   REM * COPYRIGHT (C) 1985 *
50   REM * BY MICROSPARC, INC *
60   REM * CONCORD, MA  01742 *
70   REM **********************
80   TEXT : HOME : CLEAR
90   VTAB 23: PRINT "**  COPYRIGHT 1985 BY MICROSPARC,INC  **";
100  C$(0) = "":C$(1) = "$":C$(2) = "%":PR = 1: SPEED= 200
110   VTAB 2: HTAB 10: INVERSE : PRINT " STATUS ";: NORMAL :
      PRINT " BY PAUL RAYMER": PRINT
120   PRINT : PRINT "THIS PROGRAM STARTS AT "; PEEK (103) +  PEEK
      (104) * 256
130   PRINT : PRINT "THIS PROGRAM ENDS AT "; PEEK (175) +  PEEK
      (176) * 256
140   PRINT : PRINT "THE PROGRAM LENGTH IS ";( PEEK (175) +  PEEK
      (176) * 256) - ( PEEK (103) +  PEEK (104) * 256)
```

```
150  POKE 768, PEEK (129): POKE 769, PEEK (130):V1 =  PEEK
     (768):V2 =  PEEK (769):VT = (V1 > 127) + (V2 > 127)
160  PRINT : PRINT "MOST RECENTLY USED VARIABLE IS (" CHR$ (V1)
     CHR$ (V2)C$(VT)")"
170  PRINT : PRINT "SPEED = ";256 -  PEEK (241)
180  PRINT : PRINT "DISK VOLUME "; PEEK (46017);" BOOTED IN SLOT
     "; PEEK (43626);"/DRIVE "; PEEK (43624);
190  IF  PEEK (64435) = 6 THEN  PRINT : PRINT "MEMORY SIZE IS AT
     LEAST 64K": GOTO 210
200  PRINT : PRINT "MEMORY SIZE IS ";( PEEK (978) + 35) / 4;"K"
210  VTAB 19: HTAB 1: PRINT "PADDLES SET (0)=       (1)=    "
220  VTAB 19: HTAB 17: PRINT  PDL (0): VTAB 19: HTAB 27: PRINT
     PDL (1)
230  PRINT : HTAB 8: INVERSE : PRINT "PRESS SPACE BAR TO END":
     NORMAL
240  IF  PEEK ( - 16384) = 160 GOTO 260
250  POKE  - 16336,0: GOTO 210
260  SPEED= 255: TEXT
```

Vigilant FID

Convert the FID program from your DOS 3.3 System Master to a more convenient version that is available from DOS with a simple FID command. The new version occupies the RAM card area of memory in an Apple II Plus, or a IIe, IIc or IIGS.

by Donald W. Miller, Jr., M.D.

Any Apple owner who has transferred text, binary or Applesoft files from one disk to another has surely discovered the FID program on the DOS 3.3 System Master. FID is a versatile utility program that allows you to copy, delete, lock, unlock, and catalog DOS 3.3 files. Sometimes FID is less convenient than it should be. For instance, to initialize a disk or check the contents of a particular file, you have to exit FID, and then later reload it to use it again.

The programs presented here provide you with a version of FID that resides in the RAM card area of memory (FID.RC in a 64K Apple II Plus, or a IIe, IIc or IIGS. This version incorporates the enhancements presented by Joe Humphrey in FID Plus (Reprinted elsewhere in this book), and, best of all, it is available with a simple FID command from Applesoft.

USING FID.CONVERTER AND FID.HELLO

When you run FID.CONVERTER (**Listing 62**), it loads FID from your DOS 3.3 System Master, and then stores the converted FID.RC on another disk. You have about four minutes to remove the System Master and replace it with the disk that is to receive FID.RC.

FID.RC must be installed by FID.HELLO (**Listing 63**), so be sure to save them both on the same disk. If you want the disk to boot with FID.RC in place, delete any Hello program you may already have on the disk, and rename FID.HELLO with the command:

RENAME FID.HELLO,HELLO

USING FID.RC

Once FID.RC is installed, your system can do just about anything it could before you installed FID.RC. However, you can't use Integer BASIC or any other program that uses the RAM card area of memory, and you can't use the FP command. The command FID runs FID.RC. Except for the FID Plus enhancements, FID.RC works just like the original FID.

The FID Plus enhancements are:

1. Use letters instead of numbers to select choices from the menu.
2. Where one-character responses are expected, a carriage return is not required.
3. The COPY command has been changed to MOVE.
4. Instead of the equal sign (=), the wildcard character is the asterisk (*).

Select the Q option from the menu, and you're back in Applesoft. Any program you had there will be lost, but you don't have to reboot the system.

ENTERING THE PROGRAMS

To key in the programs, first type in FID.CONVERTER (**Listing 62**), and save it with the command:

SAVE FID.CONVERTER

Then type in FID.HELLO (**Listing 63**), and save it with the command:

SAVE FID.HELLO

THE CONVERSION PROCESS

FID is located at $803 and is approximately 4,700 bytes long. Using the Monitor's disassemble command, it's easy to see that the information from $803-$1317 is made up of 6502 instructions. At first, the remaining bytes appear to be random, disorganized numbers. After some analysis, though, the information in **Table 7** can be deduced. There are three ways to convert a program such as FID to run at another location:

1. Disassemble the program and change each position-dependent reference according to the new location of the program.
2. Use a symbolic disassembler, such as the Sourceror program that comes with the Big Mac assembler.
3. Write a program to perform the task.

TABLE 7: FID Program Functions

Function	Address
$13CA-$13DB*	Table of subroutines
$13DC-$13E7*	Table of buffers
$13E8-$1449*	Offset table for ensuing text
$144A-$18F8	Text
$18F9-$190D*	File Manager parameter
$193C-$194C*	Input/output control block (IOB)
$194D	Device characteristics table
$1AF1*	Copy buffer start

*The addresses marked with asterisks will need to be changed when FID is relocated.

Since the first procedure promised to be tedious, and the second required that FID be broken into two parts, I chose the third option. This resulted in a conversion program that is specifically designed for FID. However, in the Modifications section below, I suggest some changes to make it more generally applicable.

The process turned out to be more than a simple relocation. Since FID relies on certain Monitor routines and the new location of FID conflicts with the Monitor in ROM, the appropriate Monitor routines had to be copied into the RAM card. This avoids switching back and forth between the RAM card and ROM. To avoid overwriting the copied Monitor routines, FID's copy buffer was moved from just after FID to $951.

THE INTERFACE

INIT seems to be the favorite command to discard when room is needed for a new DOS command. However, an intact INIT command can be complementary to FID in file management. A more logical candidate is the INT command, since loading the RAM card with FID eliminates the possibility of using Integer BASIC. (Besides, INT is three letters long, and so is FID.) The FP command also has to be deactivated to avoid any unexpected problems. The location of the INT command handler is $A59E. The original DOS code is overwritten with the following:

```
JSR $FB39       SET THE TEXT PAGE
LDA $C083       WRITE-ENABLE THE RAM CARD
LDA $C083
JSR $D003       RUN THE FID PROGRAM
LDA $C081       GO BACK TO ROM
JMP $3D3        COLD START DOS
```

The actual command (FID) is POKEd over INT in DOS's command table. The exit locations in FID also had to be directed to the FID command handler. This was done simply by placing an RTS command where FID had tried to JMP to $3D3.

HOW THE PROGRAMS WORK

FID.CONVERTER

The FID.CONVERTER program (**Listing 62**) finds all the bytes that have to be changed, and changes them. It goes through memory, one byte at a time, and evaluates each byte to determine whether the following bytes have to be changed.

In 6502 machine language programs, there are one-byte, two-byte, and three-byte instructions. None of the one-byte instructions has to be changed. Most of the three-byte instructions have to be changed, since these usually contain references to addresses within the program. Some of the two-byte instructions must be changed. The work of FID.CONVERTER is to skip the one-byte instructions, to determine which two-byte instructions to skip, and to make the necessary changes in the remaining two- and three-byte instructions.

First, FID.CONVERTER reads into the array (O(x)) all of the opcodes that are three-byte instructions (**line 280**). A second array (S(x)) is filled with the opcodes that are two-byte instructions (**line 290**). With these lists at hand, the program BLOADs FID at location $1803 ($1000 higher than its normal location) and PEEKs each location.

At **line 50** the memory byte is checked against the S(x) array. If there is a match, we know the following byte doesn't have to be checked, and rather than have this byte evaluated against the O(x) array, the counter is simply advanced. If the byte in question simply refers to an address, (i.e., it is in the O(x) array) the program then looks at the high byte (**line 100**).

In **line 110**, a check is made to see if the address refers to a location within the FID program rather than to a Monitor routine. **Line 120** adjusts the byte if it is in the variable storage area of FID. If it isn't, **line 130** adjusts the value of the byte so FID will be at home in the RAM card. The remaining lines (**lines 140-190**) change the assorted locations in FID, as previously described.

FID.HELLO

The FID.HELLO program listing is well-documented. Of note is the S.H. Lam (*Call-A.P.P.L.E.*) routine used to POKE in binary data (**lines 60, 70 and 110**). Also, the Monitor MOVE routine is taken from my article "Escape from the Motherboard" in the March 1983 issue of *Call-A.P.P.L.E.* Just for fun, in **line 100** the cursor routine can be modified for a nonflashing cursor (NFC) on the Apple II or II Plus. Just remove the REM at the beginning of the line.

MODIFICATIONS

If all of the FID-specific program lines were removed, FID.CONVERTER could be changed into an all-purpose machine language program relocater. All the opcodes could be placed in the appropriate array, and although slow, an accurate relocation could be done. However, the trick is to know which opcodes to include and which to exclude. Also, a working knowledge of machine language programming is needed to know what portions of the program to change. Anyone with this degree of skill could probably write a machine language program to do it all in a flash.

CAUTIONS

Disks INITialized with FID.RC resident will contain a modified DOS. Specifically, the INT and FP commands are disabled. An attempt to call FID when FID.RC has not been installed will land you in the Monitor. Also, if you have a program in memory and you then use FID, don't expect your program to be there when you return. FID.RC uses the memory where Applesoft programs are stored and destroys normal program pointers.

LISTING 62: FID.CONVERTER

```
1   REM **********************
2   REM * FID.CONVERTER      *
3   REM * BY DONALD MILLER   *
4   REM * COPYRIGHT (C) 1985 *
5   REM * BY MICROSPARC, INC *
6   REM * CONCORD, MA  01742 *
7   REM **********************
10   REM  FID -> FID.RC CONVERTER
20   GOTO 260
30 M = M + 1:F = 0: IF M > 8983 THEN 140
40 V =  PEEK (M)
50   FOR I = 1 TO 10: IF V = S(I) THEN F = 1:I = 10: GOTO 60
60   NEXT : IF F THEN M = M + 1: GOTO 30
70   FOR I = 1 TO 26: IF V = O(I) THEN F = 1:I = 26: GOTO 80
80   NEXT
90   IF  NOT F THEN 30
100 M = M + 2:Z =  PEEK (M)
110   IF Z < 8 OR Z > 30 THEN 30
120   IF Z > 25 AND Z < 30 THEN  POKE M,Z - 17: GOTO 30
130   POKE M,Z + 200: GOTO 30
140   FOR I = 9163 TO 9179 STEP 2: POKE I, PEEK (I) + 200: NEXT :
      REM  SUBROUTINE TABLE
150   POKE 9181,225: POKE 9183,10: POKE 9185,11: POKE 9187,9:
      POKE 9189,12: POKE 9191,225: REM  BUFFER TABLE
160   FOR I = 9193 TO 9289 STEP 2: POKE I, PEEK (I) + 200: NEXT :
      REM  OFFSET TABLE
170 K =  PEEK (8822) = 250: POKE 10498 + K,219: POKE 10502 +
      K,225: POKE 10504 + K,10: POKE 10506 + K,12: REM  FILE
      MANAGER PARM LIST
180   POKE 10563 + K,225: REM  IOB POKE
190   POKE 6239,96: POKE 7637,96: REM  CHANGE JMP $3D3 TO RTS
200   POKE 6840,12: REM  OMIT IF FID PLUS NOT INSTALLED
```

```
210 N = 1: FOR I = 9994 TO 10016: POKE I,128 +  ASC ( MID$
     (S$,N,1)):N = N + 1: NEXT : REM   POKE IN TITLE
220  PRINT  CHR$ (4)"BSAVE FID.RC,A$1803,L4700"
230  POKE 49281,0: POKE 49281,0: REM  WRITE TO RAM
240  PRINT  CHR$ (4)"BLOAD FID.RC,A$D003"
250  END
260  DIM O(26),S(10): HOME : PRINT : PRINT "** COPYRIGHT 1985
     MICROSPARC, INC.**": VTAB 9: PRINT "PLACE DISK WITH FID IN
     DRIVE": PRINT : PRINT "THEN HIT ANY KEY";: GET A$
270  HOME : VTAB 9: PRINT "PLEASE WAIT (APPROX 4 MINUTES)...."
280  FOR I = 1 TO 26: READ O(I): NEXT
290  FOR I = 1 TO 10: READ S(I): NEXT
300  DATA  141,142,173,76,32,189,157,185,205,204,221,172,13,140,
     29,153,174,44,217,236,57,109,238,25,206,62
310  DATA     133,162,201,240,144,105,160,208,169,176
320 S$ = "FID.RC BY D W MILLER JR": REM    23 PLACES YOUR TITLE
     HERE
330 M = 6146
340  PRINT  CHR$ (4)"BLOAD FID,A$1803"
350  GOSUB 380: REM  FID PLUS ENHANCEMENTS
360  PRINT : PRINT "INSERT DISK TO RECEIVE FID.RC"
370  GOTO 30
380  REM   FID PLUS
390 K =  PEEK (10373) = 0: FOR I = 10373 TO 10376: READ N: POKE
     I + K,N: NEXT
400  DATA  205,207,214,197
410  FOR I = 9135 TO 9161: READ N: POKE I,N: NEXT
420  DATA  205,195,211,213,204,196,210,214,209,0,210,209,0,196,
     195,204,211,213,214,0,205,00,195,210,211,209,0
430  FOR I = 6343 TO 6347: READ N: POKE I,N: NEXT
440  DATA  12,253,32,237,253
450  FOR I = 6833 TO 6860: READ N: POKE I,N: NEXT
460  DATA  76,193,251,162,11,44,162,12,32,205,10,32,12,253,141,
     0,2,170,32,237,253,32,142,253,138,162,1,96
470  FOR I = 1 TO 7: READ P: POKE P,188: POKE P + 1,10: NEXT
480  DATA  6465,6501,6541,6577,6771,7198,8101
490  POKE 6982,180: POKE 6983,10: POKE 7019,180: POKE 7020,10:
     POKE 7794,183: POKE 7795,10
500  POKE 7808,180: POKE 7809,10
510  POKE 7819,183: POKE 7820,10
520  FOR I = 1 TO 5: READ P: POKE P,170: NEXT
530  DATA 6712,6736,7393,7424,7465
540  RETURN
```

LISTING 63: FID.HELLO

```
1  REM *********************
2  REM * FID.HELLO         *
3  REM * BY DONALD MILLER  *
4  REM * COPYRIGHT (C) 1985 *
5  REM * BY MICROSPARC, INC *
6  REM * CONCORD, MA  01742 *
```

```
7  REM ***********************
10  REM  FID.RC HELLO PROGRAM
20  HOME : VTAB 9: FLASH : PRINT "INSTALLING FID.RC": NORMAL :
     PRINT : PRINT "** COPYRIGHT 1985 BY MICROSPARC, INC.**":
     PRINT : PRINT "TYPE 'FID' AT PROMPT TO ACCESS"
30  POKE 49281,0: POKE 49281,0: REM  ENABLE ROM  WRITE ENABLE
     RAM
40  POKE 43249,70: POKE 43250,73: POKE 43251,196: REM   REPLACE
     INT WITH FID COMMAND
50  POKE 43247,64: REM  DISABLE FP
60 H$ = "A59E: 20 39 FB AD 83 C0 AD 83 C0 20 03 D0 AD 81 C0 4C
     D3 03 N D7D2G": GOSUB 120: CALL  - 144: REM       FID.RC
     CONTROLLER
70 H$ = "300: A9 00 85 3C 85 42 A8 A9 FF 85 3E 85 3F A9 F8 85 43
     85 3D 20 2C FE 60 N D7D2G": GOSUB 120: CALL  - 144: REM
     MONITOR MOVE
80  PRINT  CHR$ (4)"BLOAD FID.RC,A$D003"
90  CALL 768: REM   MOVE MONITOR
100  REM      DON'T USE THESE POKES IF YOU HAVE A IIe:POKE
     64787,234: POKE 64788,234: POKE 49282,0: REM     ADD NFC
110  HOME : NEW
120  FOR I = 1 TO  LEN (H$): POKE 511 + I, ASC ( MID$ (H$,I,1))
     + 128: NEXT : POKE 72,0: RETURN
```

Eye Openers

Use this routine to create an opening iris transition from one Hi-Res picture to another.

by Iver P. Cooper

Television programs use a variety of fades and wipes to make the transition from one scene to the next. One of the most interesting of these effects is called the opening iris: an ever-widening hole appears in the center of the old image, revealing the new image. IRIS is a machine language routine that simulates this effect using the Hi-Res graphics screens.

USING THE PROGRAMS

When IRIS (**Listing 64**) is CALLed from within another program, it opens a rectangular iris on Hi-Res page 1, revealing the picture on Hi-Res page 2. Before IRIS is called, the HGR command must be issued, and the contents of the two Hi-Res pages must be set.

IRIS.DEMO (**Listing 65**) is a simple demonstration program that loads a Hi-Res picture onto page 2, BLOADs IRIS, clears page 1 to white, and repeatedly calls the IRIS routine until a key is pressed.

ENTERING THE PROGRAMS

Please refer to Appendix A for help in entering **Listing 64**. If you key it in from the Monitor, save it to disk with the command:

BSAVE IRIS,A$6000,L$DB

To key in the demonstration program, type in **Listing 65** and save it with the command:

SAVE IRIS.DEMO

HOW IRIS WORKS

To understand the method IRIS uses to switch images between the Hi-Res graphics pages, it may be helpful to examine a brief analogy. Think about the two Hi-Res pages as a two-story building under construction. The girders running one way are labeled Y=0, Y=1, and so on, and the girders that are perpendicular are labeled X=0, X=1, and so on. The top floor is Hi-Res page 2, and the bottom floor is Hi-Res page 1.

We want to wend our way as follows: starting just down and to the left of the geometric center of the top floor, take a few steps toward the top of the screen, turn right, take a few steps toward the right edge of the screen, turn right, take a few steps downscreen, turn right, take a few steps toward the left edge of the screen, turn right, and so on — spiraling out until we reach the edge of the building. We then want to recognize that we have reached the edge, and stop.

Assume that each girder intersection is marked with a number. With each step, we want to read off the intersection number for the benefit of a friend on the floor below, who will mark it on the corresponding intersection there.

For the purposes of this program, I thought of the Hi-Res screen as being divided into 40x40 (mixed text and graphics mode) Lo-Res blocks. Each Lo-Res block is four pixels high and seven pixels wide. If we start at the block just to the lower-left of the exact center of the screen, and move upscreen one block, turn and move one block, turn and move two blocks, turn and move two blocks, turn and move three blocks, turn and move three blocks, turn and move four blocks, and so on, we will eventually travel down to the lower-right block of the mixed mode screen.

In the source code for IRIS, **lines 29-32** position the program's internal graphics cursor in the proper starting point. The locations UPBY, DNBY, RTMARG and LFMARG control how far it moves. These locations change just before a turn, so that a widening iris effect is achieved.

The internal cursor consists of a two-byte pointer at $26,$27 to the memory location for the left edge of the cursor's row; a column block number at $E5, in the range 0-39; and a point-within-column indicator at $30. Block 0,40 on page 2 has memory address $4250. The subroutine CHECK, which is called after every cursor movement in the DN segment of the main routine, compares the value of the left edge pointer to the address of that block.

A second subroutine, called TRNSFR, actually moves the contents of the graphics screen. TRNSFR:

1. Gets the value stored in the appropriate row and column on page 2.
2. Saves this value on the stack.
3. Moves the pointer to the corresponding position on page 1.
4. Gets the value saved on the stack and places it into page 1.
5. Restores the left edge pointer to page 2.

The actual movement of the internal cursor is accomplished by CALLs to the ROM subroutines INCRY and DECRY, and by incrementing or decrementing the value at $E5.

MODIFICATIONS

To have a smaller iris opening on a corner of the screen, change the starting point in **lines 29-32**, the initial values of LFMARG and RTMARG, and the comparison values in CHECK. To have the program overlay rather than erase the information originally on page 1 with whatever is on page 2, insert the instruction ORA ($26),Y before the instruction STA ($26),Y.

LISTING 64: IRIS

```
 0                        ;
 1                        ;        IRIS
 2                        ;  BY IVER P. COOPER
 3                        ; COPYRIGHT (C) 1985
 4                        ; BY MICROSPARC, INC
 5                        ; CONCORD, MA  01742
 6                        ;
 7                                 ORG $6000
 8                        PAGE     EQU $E6
 9                        PAGE2    EQU $40
10                        DECRY    EQU $F4D5
11                        INCRY    EQU $F504
12                        COLUMN   EQU $E5
13                        UPBY     EQU $FC
14                        DNBY     EQU $FD
15                        RTMARG   EQU $FE
16                        LFMARG   EQU $FF
17                        HPOSN    EQU $F411
18                        ;
19  6000  A9 08           INIT     LDA #8
20  6002  85 FC                    STA UPBY
21  6004  A9 0C                    LDA #12
22  6006  85 FD                    STA DNBY
23  6008  A9 14                    LDA #20
24  600A  85 FE                    STA RTMARG
25  600C  A9 12                    LDA #18
26  600E  85 FF                    STA LFMARG
27  6010  A9 40                    LDA #PAGE2
```

```
6012  85 E6             STA PAGE
6014  A9 53             LDA #83
6016  A0 00             LDY #0
6018  A2 85             LDX #133
601A  20 11 F4          JSR HPOSN
601D  20 B5 60          JSR TRNSFR
6020  A6 FC     UP      LDX UPBY
6022  20 D5 F4  UP2     JSR DECRY
6025  20 B5 60          JSR TRNSFR
6028  CA                DEX
6029  D0 F7             BNE UP2
602B  A5 FC             LDA UPBY
602D  18                CLC
602E  69 08             ADC #8
6030  85 FC             STA UPBY
                ;
6032  E6 E5     RT      INC $E5
6034  A4 E5             LDY $E5
6036  C4 FE             CPY RTMARG
6038  F0 0A             BEQ RT2
603A  90 08             BCC RT2
603C  C6 E5             DEC $E5
603E  88                DEY
603F  E6 FE             INC RTMARG
6041  4C 68 60          JMP DN
6044  20 B5 60  RT2     JSR TRNSFR
6047  A5 26             LDA $26
6049  48                PHA
604A  A5 27             LDA $27
604C  48                PHA
604D  20 04 F5          JSR INCRY
6050  20 B5 60          JSR TRNSFR
6053  20 04 F5          JSR INCRY
6056  20 B5 60          JSR TRNSFR
6059  20 04 F5          JSR INCRY
605C  20 B5 60          JSR TRNSFR
605F  68                PLA
6060  85 27             STA $27
6062  68                PLA
6063  85 26             STA $26
6065  4C 32 60          JMP RT
6068  A6 FD     DN      LDX DNBY
606A  20 04 F5  DN2     JSR INCRY
606D  20 CC 60          JSR CHECK
6070  20 B5 60          JSR TRNSFR
6073  CA                DEX
6074  D0 F4             BNE DN2
6076  A5 FD             LDA DNBY
6078  18                CLC
6079  69 08             ADC #8
607B  85 FD             STA DNBY
                ;
607D  C6 E5     LF      DEC $E5
607F  A4 E5             LDY $E5
6081  30 06             BMI LFFLP
6083  C4 FF             CPY LFMARG
6085  F0 0A             BEQ LF3
6087  B0 08             BCS LF3
```

```
6089  E6 E5     LFFLP   INC $E5
608B  C8                INY
608C  C6 FF             DEC LFMARG
608E  4C 20 60          JMP UP
6091  20 B5 60  LF3     JSR TRNSFR
6094  A5 26             LDA $26
6096  48                PHA
6097  A5 27             LDA $27
6099  48                PHA
609A  20 D5 F4          JSR DECRY
609D  20 B5 60          JSR TRNSFR
60A0  20 D5 F4          JSR DECRY
60A3  20 B5 60          JSR TRNSFR
60A6  20 D5 F4          JSR DECRY
60A9  20 B5 60          JSR TRNSFR
60AC  68                PLA
60AD  85 27             STA $27
60AF  68                PLA
60B0  85 26             STA $26
60B2  4C 7D 60          JMP LF
                ;
60B5  A4 E5     TRNSFR  LDY $E5
60B7  B1 26             LDA ($26),Y
60B9  48                PHA
60BA  38                SEC
60BB  A5 27             LDA $27
60BD  E9 20             SBC #$20
60BF  85 27             STA $27
60C1  68                PLA
60C2  91 26             STA ($26),Y
60C4  A5 27             LDA $27
60C6  18                CLC
60C7  69 20             ADC #$20
60C9  85 27             STA $27
60CB  60                RTS
                ;
60CC  A5 26     CHECK   LDA $26
60CE  C9 50             CMP #$50
60D0  D0 08             BNE CHECKZ
60D2  A5 27             LDA $27
60D4  C9 42             CMP #$42
60D6  D0 02             BNE CHECKZ
60D8  68                PLA
60D9  68                PLA
60DA  60        CHECKZ  RTS

000  ERRORS

6000  HEX START OF OBJECT
60DA  HEX END OF OBJECT
00DB  HEX LENGTH OF OBJECT
955A  HEX END OF SYMBOLS
```

LISTING 65: IRIS.DEMO

```
10  REM ***********************
20  REM *      IRIS.DEMO      *
30  REM * BY IVER P. COOPER   *
40  REM * COPYRIGHT (C) 1985 *
50  REM * BY MICROSPARC, INC *
60  REM * CONCORD, MA  01742 *
70  REM ***********************
80  PRINT  CHR$ (4);"BLOAD IRIS"
90  HOME : PRINT "ENTER THE NAME OF THE HI-RES PICTURE    FILE
     (TRY 'DEMO')";: INPUT NA$
100  PRINT  CHR$ (4);"BLOAD";NA$;",A$4000"
110  HGR : HOME : VTAB 22: PRINT "** COPYRIGHT 1985 BY
     MICROSPARC, INC.**"
120  HCOLOR= 3: REM  WHITE
130  HPLOT 0,0
140  CALL 62454: FOR I = 1 TO 1000: NEXT : REM  CLEAR SCREEN TO
     WHITE AND PAUSE
150  CALL 24576: REM   CALL IRIS ($6000)
160  IF  PEEK ( - 16384) < 128 THEN  FOR I = 1 TO 1000: NEXT :
     GOTO 110: REM LOOP UNTIL KEY PRESS
170  POKE  - 16368,0: TEXT : HOME : END
```

Imagewriter Screen Dump

Learn how to use the Imagewriter Tool Kit Hi-Res screen dump routine from your own program. You can dump either Hi-Res page in four different modes.

by Gerald Blalock

The Imagewriter printer is often sold as part of a package with the Apple IIe or IIc. Its Tool Kit disk includes a menu-driven graphics screen dump program that handles a screen dump of Hi-Res page 1 in normal or inverse text, and regular or double-size fonts. Those are the functions that the documentation describes, however, the Tool Kit has some hidden talents.

You can bypass the menus and call the routine from other program and in immediate mode. Furthermore, dumps can be made from page 2. Immediate mode access is as simple as a BLOAD, two POKEs, and a CALL for DOS 3.3, or three POKEs and two CALLs for ProDOS. Since the DOS 3.3 and ProDOS Imagewriter Tool Kit screen dump programs differ significantly, they are discussed separately.

DOS 3.3 IMAGEWRITER DUMP

The Applesoft Hello program on the DOS 3.3 Imagewriter Tool Kit disk CALLs a machine language program, GF, which is located at $9000. A little experimentation and disassembly of the GF program reveals eight different modes that are set by a variable I will call XFEROPT.

Values zero through three control screen dumps from Hi-Res page 1 and determine both the size of the image and whether it is printed in inverse. Values four through seven of XFEROFT are the same as the first four, except that they apply to Hi-Res page 2. See **Table 8** for the values of XFEROPT and their corresponding modes.

TABLE 8: Values of XFEROPT and Corresponding Modes

	Number to POKE in Location 7	
Option	**Page 1**	**Page 2**
Normal	0	4
Inverse	1	5
Double size Normal	2	6
Double size Inverse	3	7

A general-purpose screen dump program that lets you load a picture into either page and dump it in any of the four modes is presented in **Listing 66**. (The DOS 3.3 GF program does not work on the IIc or IIGS. Instead, you must use the ProDOS version.) To incorporate the screen dump routines within your own programs, use the following procedure:

1. BLOAD GF from your Imagewriter Tool Kit disk.
2. BLOAD *picture*,A$2000 for Hi-Res page 1, or BLOAD *picture*,A$4000 for Hi-Res page 2.
3. POKE 6,PSLOT:POKE 7, XFEROPT (PSLOT is the slot holding your printer interface card, and XFEROPT is the value obtained from **Table 8**.)
4. PR# PSLOT: PRINT CHR$(27);CHR$(78)
5. CALL 36864

PRODOS IMAGEWRITER DUMP

The ProDOS Imagewriter screen dump program works on the Apple II Plus, IIe, IIc and IIGS. The machine language program, named GRAF.0 on the ProDOS Imagewriter Tool Kit disk, is essentially the same as GF, except for the initial setup portion. The result is that three POKEs and two CALLs are required to use GRAF.0 from the immediate mode or from within an Applesoft program. Unlike the DOS 3.3 version,the ProDOS version requires no printer setup.

Listing 67 is a general-purpose screen dump that uses the ProDOS program GRAF.0. Use the following general procedure:

1. Set HIMEM: 36864 to protect the GRAF.0 code.
2. BLOAD GRAF.0
3. BLOAD *picture*,A$2000 for Hi-Res page 1, or BLOAD *picture*,A$4000 for Hi-Res page 2.
4. POKE 6,PSLOT: POKE 252,16 * PSLOT: POKE 7, XFEROPT (PSLOT is the number of the slot holding the printer interface card, and XFEROPT is the mode number from **Table 8**.)
5. CALL 38636
6. CALL 38156

ENTERING THE PROGRAMS

If you have the DOS 3.3 version of the Imagewriter Tool Kit disk, enter the program shown in **Listing 66** and save it on a disk that contains the file GF with the command:

SAVE DOS3.3.DUMP

If you have the ProDOS version of the Imagewriter Tool Kit, enter the program shown in **Listing 67** and save it on a disk that contains the file GRAF.0 with the command:

SAVE PRODOS.DUMP

Note: Apple Computer, Inc. is no longer distributing the DOS 3.3 Imagewriter Tool Kit on which the GF file needed for DOS3.3.DUMP is supplied. However, the GF file is included on the More Apple Secrets disk; see the bound-in card at the end of this book for ordering information.

LISTING 66: DOS3.3.DUMP

```
10  REM  **********************
20  REM  *     DOS3.3.DUMP     *
30  REM  *  BY GERALD BLALOCK *
40  REM  * COPYRIGHT (C) 1985 *
50  REM  * BY MICROSPARC, INC *
60  REM  * CONCORD, MA  01742 *
70  REM  **********************
80  REM    HI-RES DUMP USING IMAGEWRITER PRINTER
90  REM    THE FILE 'GF' AND YOUR PICTURE FILE MUST BE ON THE
     SAME DISK
100  HOME : HIMEM: 36864
110 D$ =  CHR$ (4): REM   CTRL-D
120 PSLOT = 1: REM   PRINTER SLOT
130  PRINT D$;"BLOAD GF"
140  POKE 6,PSLOT
```

```
150  PRINT "NAME OF PICTURE FILE (? FOR CATALOG)": INPUT ":
     ";NAME$: IF NAME$ = "?" THEN  PRINT D$"CATALOG": GET Z$:
     PRINT : GOTO 150
160  INPUT "WHICH PAGE (1 OR 2): ";P$
170 P$ =  LEFT$ (P$,1): IF P$ <  > "1" AND P$ <  > "2" GOTO 160
180  PRINT D$;"BLOAD "NAME$",A$"2000 + 2000 * (P$ = "2")
190  PRINT "PRINT MODES:": PRINT "  0  SINGLE NORMAL": PRINT "
     1  SINGLE INVERSE ": PRINT "  2  DOUBLE NORMAL": PRINT "  3
     DOUBLE INVERSE "
200  INPUT "PRINT MODE? ";XFEROPT
210  POKE 7,XFEROPT + 4 * (P$ = "2")
220  PRINT D$;"PR#";PSLOT: PRINT  CHR$ (27) +  CHR$ (78): REM
     TURN ON GRAPHICS MODE
230  CALL 36864: REM   PRINT IT
240  PRINT : PRINT D$;"PR#0": REM   ALL DONE!
```

LISTING 67: PRODOS.DUMP

```
10  REM  **********************
20  REM  *     PRODOS.DUMP    *
30  REM  *  BY GERALD BLALOCK *
40  REM  * COPYRIGHT (C) 1985 *
50  REM  * BY MICROSPARC, INC *
60  REM  * CONCORD, MA  01742 *
70  REM  **********************
80  REM    HI-RES DUMP USING IMAGEWRITER PRINTER
90  REM    THE FILE 'GRAF.0' AND YOUR PICTURE FILE MUST BE ON
     THE SAME DISK
100  HOME : HIMEM: 36864
110 D$ =  CHR$ (4): REM   CTRL-D
120 PSLOT = 1: REM   PRINTER SLOT
130  PRINT D$;"BLOAD GRAF.0"
140  POKE 6,PSLOT: POKE 252,16 * PSLOT
150  PRINT "NAME OF PICTURE FILE (? FOR CATALOG) ": INPUT ":
     ";NAME$: IF NAME$ = "?" THEN  PRINT D$"CAT": GET Z$: PRINT
     : GOTO 150
160  INPUT "WHICH PAGE (1 OR 2): ";P$
170 P$ =  LEFT$ (P$,1): IF P$ <  > "1" AND P$ <  > "2" GOTO 160
180  PRINT D$;"BLOAD "NAME$",A$"2000 + 2000 * (P$ = "2")
190  PRINT "PRINT MODES:": PRINT "  0  SINGLE NORMAL": PRINT "
     1  SINGLE INVERSE ": PRINT "  2  DOUBLE NORMAL": PRINT "  3
     DOUBLE INVERSE "
200  INPUT "PRINT MODE? ";XFEROPT
210  POKE 7,XFEROPT + 4 * (P$ = "2")
220  CALL 38636: REM  SET UP PRINTER
230  CALL 38156: REM  PRINT IT
```

APPENDIX A

Entering More Apple Secrets Program Listings

More Apple Secrets includes programs written in Applesoft BASIC and machine language. Both types of programs can be entered directly into your Apple, without the use of additional software. This appendix presents some of the basics of program entry for those who are new to Apple computing. While this short summary is no substitute for Apple's manuals, it should be enough to get you started on More Apple Secrets program listings.

A QUICK OVERVIEW OF THE APPLE

When you first switch on your Apple, make sure that either the DOS 3.3 System Master disk or the ProDOS System disk is in the disk drive. If you use the ProDOS System disk, you will need to quit the startup program. You will the see a square bracket (]) character, called a prompt. The square bracket prompt tells you that you can do one of three things:

1. Enter commands in the disk command language (e.g., CATALOG).

2. Enter commands in Apple's version of the BASIC language, Applesoft BASIC (e.g., PRINT 36+42).

3. Type in Applesoft BASIC program lines (e.g., 10 INPUT K).

To type in programs from More Apple Secrets, you may need to do all three.

ENTERING AN APPLESOFT BASIC PROGRAM

Before entering a program listing from More Apple Secrets, you should first thoroughly read the text that describes the program. You may not understand all of the explanations the first time through, but be on the lookout for any special directions for typing the program. You should also be sure to have a formatted disk ready so that you can save your work.

All BASIC programs consist of a sequence of program lines. Each program line begins with a number and is followed by one or more program statements separated by colons. For example:

```
20 FOR J = 1 TO 5:PRINT CHR$ (7):NEXT J
```

To enter a program, begin with the first numbered line and type it in exactly as it appears (including the line number itself). Though the program line may span several printed lines in the listing, do not press the Return key until you reach the next line number. Then begin the process again with the next line number. When you reach the end of the program, save your work on the disk by typing the command SAVE followed by the name of the program. That's all there is to it!

Let's try a sample program. To enter the program BELLS shown in Listing 1, follow this sequence:

1. Type the word NEW and press Return to clear memory of any old programs. (Make sure the Caps Lock key is down if you are using an unenhanced Apple //e.)

2. Type line 10 exactly as it appears, but do not press Return until you have typed the last word in the line, "BELL".

3. Repeat this procedure with lines 20 and 30.

4. With an initialized disk in the drive, type SAVE BELLS and press Return to save your program on the disk.

5. Since the program is now in memory, you may just type RUN and press Return to start it. If you erase it from memory by running a different program or by turning off your computer, you may put it back into memory and start it again by typing the command RUN BELLS and pressing Return.

LISTING 1

```
10 REM RING THE BELL
20 FOR J = 1 TO 5: PRINT CHR$ (7): NEXT J
30 END
```

A FEW TIPS

The following tips may make your work a little easier:

1. If you make a mistake while typing, use the Left-Arrow key to go back and correct it, and the Right-Arrow key to "retype" the remainder of the line before pressing Return. If you have already pressed Return before you catch your error, simply retype the entire line (number and all) and the new version will take the place of the old. (The use of an Applesoft line editor like MicroSPARC's GALE can eliminate much of this work.)

2. Be particularly careful when typing in statements that contain the reserved word DATA. Typos in other lines will probably show up as syntax errors when the program is finally run, but those in DATA statements may not.

3. Save the program to disk periodically as you go along to minimize the effect of an accidental power loss.

4. Don't try to make your own modifications to the program until you have typed it in as published and have run it successfully. This will make it easier to debug in case you have made typing errors.

5. If the program does not seem to run correctly, it may be helpful to temporarily remove any ONERR statements. This will allow you to see error messages suppressed by ONERR.

6. If you're certain that you have typed the program correctly, but you still can't get it working, call MicroSPARC's Technical Support Staff at (617) 371-1660 for assistance.

ENTERING MACHINE LANGUAGE PROGRAMS

Both BASIC and the disk command language are powerful languages that interpret English-like words. Your Apple can also understand a much lower-level language, called machine language. Since this is the Apple's "native tongue," machine language programs perform much more quickly than those written in BASIC.

Often, a program called an assembler is used to help create machine language programs. An assembler first allows the programmer to write an assembly language program and then translates this program into machine language before it is run. Though you may not have an assembler, you will still be able to enter and use the machine language from More Apple Secrets listings. The advantage of an assembler is that it allows you to easily modify the program, or to "borrow" a programming technique. Unless otherwise indicated, all assembly language programs in More Apple Secrets were produced using The Assembler from MicroSPARC, Inc.

If you don't own an assembler, you will need to enter machine language programs directly into the Apple's memory through what is called the System Monitor (not to be confused with your video monitor). To reach this level from the disk/BASIC level (indicated by the ']' prompt), you simply type CALL -151 and press Return. You will then see an asterisk (*), which is the prompt for the System Monitor. While you can use many commands at this level, the only one you will need to enter More Apple Secrets listings looks like this example:

```
300:A2 05 20 DD FB CA F0 03 4C 02 03 60
```

In this command, the "300" specifies a memory location in your Apple and the colon tells the Apple to put the following number (A2, a number in base 16) into that location. The numbers following the first (05 through 60) are put into subsequent memory locations. (Of course, you would press Return at the end of the line.) Though you don't need to understand base 16 (or hexadecimal) numbers, you should know that all machine language numbers are given in hexadecimal notation.

A SAMPLE MACHINE LANGUAGE PROGRAM

Let's follow a short example of entering a machine language program. Listing 2 shows the contents of a portion of the Apple's memory, often called a "hex dump." The number to the left of the hyphen is a memory location's "address," and the numbers to the right are the contents of that and subsequent memory locations.

LISTING 2

```
0300- A2 05 20 DD FB CA F0 03
0308- 4C 02 03 60
```

Listing 3 shows the assembly language which was used to create the machine language program shown. Notice that the numbers in the left-hand columns look very similar to those in Listing 2. They are, in fact, the same set of memory addresses and their program contents in a different format. All of the columns on the right are assembly language instructions and comments. While other assemblers use slightly different formats, you will always be able to find the two columns which contain the addresses and contents of memory.

LISTING 3

```
                    ; RINGER PROGRAM
                             ORG  $300
                    BELL     EQU  $FBDD
  0300   A2 05               LDX  #$5
  0302   20 DD FB   LOOP     JSR  BELL
  0305   CA                  DEX
  0306   F0 03               BEQ  END
  0308   4C 02 03            JMP  LOOP
  030B   60         END      RTS
```

To enter the machine language listings, you just type in the addresses and their contents as follows:

1. Type CALL -151 and press Return to get into the System Monitor. You should now have an asterisk (*) prompt.

2. Type the first memory address shown, a colon (instead of the hyphen shown in the listing), and the memory contents. If you were using a listing similar to Listing 2, you would type:

```
300:A2 05 20 DD FB CA F0 03
308:4C 02 03 60
```

   If you were using an assembler listing like that in Listing 3, you would type:

```
300:A2 05
302:20 DD FB
305:CA
306:F0 03
308:4C 02 03
30B:60
```

   Be sure that you do not put a space between the colon and the first pair of hexadecimal digits, but that you do put spaces between subsequent pairs. Also, remember to press Return after each line. You may actually type up to 85 pairs of digits after each colon, but it is easier to follow the listing as published for your first time through. When you finish typing the program, it can be verified by typing the starting address and pressing Return until all of the code is listed. If your code does not show the values listed, retype the incorrect line.

3. When you have entered the entire listing, press Control-C and then Return to get back to the disk/BASIC level indicated by the ']' prompt. This is accomplished by pressing the C key while holding down the Control key, and then pressing the Return key.

4. While BASIC programs always start in the same place in the Apple's memory, and thus can simply be saved with the SAVE command, machine language programs can start at various places in memory. For this reason, the command to save a machine language program (BSAVE) must include the starting address (A) and the length (L) of the program being saved. For the program above, the command:

```
BSAVE RINGER,A$300,L$C
```

   would be used. (The dollar sign ($) signifies that the number is given in hexadecimal notation.) Directions for saving machine language files (with the correct address and length) are included in the text accompanying these programs.

You can now run this program by typing BRUN RINGER. (The address and length are only necessary for the BSAVE command.) You can also run this program from the disk/BASIC level (after you have BLOADed it into memory) with a CALL statement followed by the decimal equivalent of the starting address. In this case, CALL 768 can be used to run the program since 768 is the decimal form of the number $300.

Sometimes a machine language listing is not a program at all, but is merely a table of data (such as a Hi-Res graphics shape table). In these cases, the memory addresses and their contents should be typed in as described above, but you should not attempt to BRUN the file you have saved. You will be able to determine whether the machine language listing is a program or a data table by reading the accompanying text.

MORE HELP

Program editors can be used to help speed the entry and editing of More Apple Secrets programs. MicroSPARC, Inc., the publisher of More Apple Secrets and Nibble Magazine, also publishes two program editors, GALE and MLE.

GALE (Global Applesoft Line Editor) offers screen oriented editing of Applesoft program lines, global search and replace of any program text, auto line numbering, variable cross-referencing, renumbering, user-definable macro functions and much more.

MLE (Machine Language Editor) will help you enter and edit machine language listings without using the Monitor. With MLE, you can enter machine language code, delete or insert code to correct typing mistakes and save your work for later editing.

The Assembler is a complete editor and assembler system that can be used to directly enter and assemble source code. It is available in both DOS 3.3 and ProDOS versions.

To order GALE, MLE, The Assembler or a subscription to Nibble Magazine, use the convenient bound-in card at the back of this book. Phone orders are accepted with Master Card or VISA – call (617) 371-1660.

The Key to Hidden Apple Treasures

All of the programs from More Apple Secrets are available on disk! Order them using one of the convenient postage-paid cards below. You can also order The MicroSPARC Assembler, GALE and MLE, three super tools for typing and editing programs. And if you're not yet a subscriber to Nibble Magazine and you like More Apple Secrets, subscribe today to get more of the same top-quality programs and tips — every month!

Please send me:

☐ More Apple Secrets book (buy one for a friend!) — $19.95 plus $1.75 shipping.*
☐ More Apple Secrets disk (DOS 3.3; can be converted to ProDOS) — $10.00 plus $1.50 shipping.*

☐ Apple Secrets book (first book in the Apple Secrets series) — $19.95 plus $1.75 shipping.*
☐ Apple Secrets disk — $10.00 plus $1.50 shipping.*

☐ The MicroSPARC Assembler ☐ DOS 3.3 version ☐ ProDOS version (please specify one) — $49.95 postpaid.
☐ GALE (Global Applesoft Line Editor) — $49.95 postpaid.
☐ MLE (Machine Language Editor) — $29.95 plus $1.50 shipping.*

☐ Please enter my subscription to Nibble Magazine. I've enclosed $26.95 for 12 issues (U.S. only; see below for foreign rates).

Name Address

City State Zip Code

☐ I've enclosed a check or money order Charge my: ☐ Visa ☐ MasterCard

Charge Card Number Expiration Date

Signature Telephone Number AP

Please send me:

☐ More Apple Secrets book (buy one for a friend!) — $19.95 plus $1.75 shipping.*
☐ More Apple Secrets disk (DOS 3.3; can be converted to ProDOS) — $10.00 plus $1.50 shipping.*

☐ Apple Secrets book (first book in the Apple Secrets series) — $19.95 plus $1.75 shipping.*
☐ Apple Secrets disk — $10.00 plus $1.50 shipping.*

☐ The MicroSPARC Assembler ☐ DOS 3.3 version ☐ ProDOS version (please specify one) — $49.95 postpaid.
☐ GALE (Global Applesoft Line Editor) — $49.95 postpaid.
☐ MLE (Machine Language Editor) — $29.95 plus $1.50 shipping.*

☐ Please enter my subscription to Nibble Magazine. I've enclosed $26.95 for 12 issues (U.S. only; see below for foreign rates).

Name Address

City State Zip Code

☐ I've enclosed a check or money order Charge my: ☐ Visa ☐ MasterCard

Charge Card Number Expiration Date

Signature Telephone Number AP

*Shipping: More Apple Secrets disk and MLE: U.S. shipments, add $1.50 per disk; outside the U.S., add $2.50 for the first disk ordered, and $1.00 for each additional disk. More Apple Secrets book: U.S. shipments, add $1.75 per book; outside U.S., add $2.75 per book for surface mail, $6.50 per book for air mail. Payments must be in U.S. funds. Mass. residents add 5% sales tax on book and software orders.

Nibble Magazine Foreign Subscription Rates: Canada Surface $34.95; Canada Air Mail $59.95; Outside U.S. and Canada Surface $39.95; Outside U.S. and Canada Air Mail $89.95. U.S. First Class Rate (for faster delivery) $51.95. All payments must be in U.S. funds drawn on a U.S. bank.

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 43 CONCORD, MA

POSTAGE WILL BE PAID BY ADDRESSEE

nibble Magazine
45 Winthrop Street
Concord, MA 01742-9990

MORE APPLE SECRETS

The Key to Hidden Apple Treasures!

Unlock *more* mysteries of your Apple. . . with *More Apple Secrets!* Your Apple II Plus, IIc, IIe and IIGS are full of hidden treasures — discover them for yourself with this exciting anthology.

More Apple Secrets is packed with over 50 of the best articles selected from the popular Tips 'n Techniques column published in Nibble Magazine, the Reference for Apple II Computing. Nibble's experts teach you their tricks for creating Mac-like text windows from Applesoft, chaining Applesoft without a chain, and speeding up programs enough to even *hear* the difference!

Want to add dozens of new colors to your Hi-Res palette, add two-voice music and sound effects to your programs, or just customize the sound of your Apple's beep? Easy! All these programming treasures and many more can be discovered in *More Apple Secrets*. In fact, it will even tell you how to hide disk data from prying eyes!

Each Apple Secret is a foolproof method for streamlining your Applesoft and machine languge programs, ranging from special programming tips to specific techniques. Most articles include subroutines you can use in your own programs. The authors give detailed, line-by-line descriptions of the programs so you can understand the programming logic. And the demonstration programs show you how to use the techniques.

More Apple Secrets is the key to hidden Apple treasures. Use it to unlock the secrets of *your* Apple!

All programs in *More Apple Secrets* are available on diskette! See the bound-in card for ordering information.

$19.95

01
0 74470 19726 3