Open-Apple

March 1985
Vol. 1, No. 2

Releasing the power to everyone.

: FLASH :

Steve Wozniak, designer of the Apple II, is no longer involved in any Apple projects, although he remains an employee on a consulting basis, according to Linda Merrill, Apple II specialist in Apple's corporate relations department. Steve is starting a home video products company, which will develop products that will allow consumers to use video cassette recorders in new ways. Wendell Sander, head hardware designer for the Apple III, has reportedly joined Steve, but that is unconfirmed.

According to the *Wall Street Journal* (Feb 7, p. 42), Wozniak told reporters that his departure was prompted in part by bitter disagreement with Apple's management over the company's direction and by frustration with the rigidity of a big corporate bureaucracy. Wozniak said the Apple II "has been ignored in the hope that it will die and go away."

Steve's new products have "nothing to do with home computers," according to Merrill. Steve has always been a vocal spokesperson for Apple II users inside Apple itself. Although he is still "officially a part of Apple", his absence on a full-time basis is a blow to those of us already concerned about the relatively minuscule development investment Apple makes in its Apple II family.

Apple's annual stockholder's meeting, held on January 23 this year, was a Macintosh circus. *Not one* of the new products announced works with the Apple II.

Apple did announce, however, that by fall it would have a peripheral card for the IBM PC that will allow connection to **AppleTalk**—its new low-cost system for hooking (non-Apple II) personal computers, printers, large memory devices, and other peripherals together.

Apple spokesperson Merrill tells us Apple is aware that many Apple II users, particularly in educational institutions, are interested in this capability and she assures us that Apple is working on it.

Meanwhile, AppleWorks was December's sales leader in retail stores, according to Infocorp, a market research firm in Cupertino. The program walked away with 17 per cent of all software sales. The much ballyhooed **Lotus 1-2-3** was in second place with just 10 per cent of the market.

Quote of the month:

I would argue that the ultimate deciding factor in computer obsolescence is software availability. The sure sign of a computer growing into obsolescence is that no new software is being developed for it. And if software is available to do what you want done with your computer, it will not be obsolete for quite some time. (By this measure, the Apple II may never be obsolete.)

The quote is from Mark Wilsdorf, editor of *AgriComp* (103 Outdoors Bldg, Columbia, MO 65201, $24/yr), a well-done magazine that covers the area where agriculture and personal computers overlap. Much of the magazine's material is based on spreadsheet templates that work with any kind of computer. If you are one of the world's unfortunate souls involved in agriculture, you should get this one, depression or not.

New enhanced ROMs for the Apple IIe have been mentioned in various press reports the last few weeks. The new ROMs reportedly make the IIe and IIc more compatible. When Apple officially introduces these new ROMs we'll have a complete report (and if it's not by March 15, the April Open-Apple may be mostly blank).

Basham fires up the Incinerator

I got a letter and disk from Bill Basham, developer of **Diversi-DOS** (arch-competitor of my own **ProntoDOS**) this month:

Your discussion of garbage collectors in the January issue inspired me to finally write one. I've enclosed the latest Diversi-DOS *update, which includes a 48K and a 64K garbageman on it.*

I hereby donate the 48K garbageman to the public domain. Since the 64K version works only with Diversi-DOS, I will retain the rights to it. There's also a source listing for the 48K garbageman for you to publish if you wish. Of course, I used the Pascal assembler (ed note: expletive deleted)*, so you may have to do some retyping and possibly add some improved comments. Feel free to make up a catchy name for it.*

My garbageman may be a little slower, but I'm confident that it is the world's smallest. That should make it ideal for publication, and I hope will make that issue a hot one.

Bill apparently wants me firmly entrenched in the newsletter business and out of the DOS-enhancement business. After looking over his program I *am* inclined to give up trying to compete with him—this is a masterpiece.

Readers of our January issue will remember a two-page spread on how to avoid garbage collection—the malady that strikes string-intensive Applesoft programs at the most inopportune times. ProDOS solves the problem by including its own high-speed garbage collector. Now DOS 3.3 users also have a high-speed garbage collector that fits entirely within DOS. As with ProDOS, however, you have to give up the ability to initialize disks.

Basham's program, hereby christened the *Incinerator,* works by making a temporary copy of the standard Applesoft garbage collection routine and modifying it. The copy is placed in the DOS "nibble buffers", a rather large memory area that gets overwritten whenever a disk is accessed. Since the disk is inaccessible while the routine is running, however, it's a nice safe place to go to burn the garbage.

The modifications cause the old garbage collector to use two new routines at critical places. These routines, as well as the code that moves and patches the old garbage collector, resides in the DOS area normally used by the *init* command. The program begins with a third section that installs the main body inside DOS.

Here's the speed comparison from our January article with *Incinerator* timings added:

Garbage collection time
in seconds

number of strings	DOS 3.3	Incinerator	ProDOS
250	5	0.7	0.1
500	19	1.5	0.2
1000	114	4.0	0.5

For those of you interested in such stuff, the complete source code for Basham's *Incinerator* follows. You can study it with the help of the comments I've written. If all you want to do is *use* it, you can type it in by hand. However, if you have a modem, software that allows you to capture files, and access to CompuServe, you can also download it from the Micronetworked Apple User Group library (GO PCS-51). It's filed in the "Apple II Hacking" section. Select *browse* from the menu that appears and enter *INCINERATOR* when you are asked for a "/key". (All of the other programs in the first three issues of ***Open-Apple*** are also now available in the MAUG library.)

Basham's original version—the one that now comes with *Diversi-DOS*—is connected to the Applesoft ampersand hook. I have modified the version published here so that it is called automatically by DOS every time a carriage return is printed. On each call, the amount of unused memory left is checked. If there is more than about 1K, control quickly returns to DOS. If there is less, collection occurs automatically.

To *force* collection, use the following command:

```
POKE 48816,0 : PRINT : POKE 48816,4
```

The complete source code follows:

If you want to type in the *Incinerator* by hand rather than downloading it from CompuServe, the easiest way is to enter the material in the gray boxes using the Monitor commands we've covered in last month's issue and in this one. Check your entries using the Monitor's L(ist) command. Then enter *BSAVE INCINERATOR, A$4000, L$1BA.* To execute the program, enter *BRUN INCINERATOR.*

```
          *------------------------------------------------------------
          *     : INCINERATOR
          *     :    Fast garbage collector for 48K DOS 3.3
          *     :
          *     : By Bill Basham, Diversified Software Research
          *     :    January 1985
          *     :
          *     : Comments and DOS hooks by Tom Weishaar
          *     :
          *     : A public domain program
          *------------------------------------------------------------

                .OR $4000
                .TF INCINERATOR

0046:          COUNTER   .EQ $46
005E:          INDEX     .EQ $5E
006D:          STREND    .EQ $6D  the lower edge of string space
006F:          FRETOP    .EQ $6F  new strings are stored from here down
0073:          MEMSIZ    .EQ $73  the upper edge of string space, HIMEM
008A:          FNCNAM    .EQ $8A
008F:          DSCLEN    .EQ $8F
009B:          LOWTR     .EQ $9B

03D0:          P3.DOSWARM .EQ $3D0   page 3 vector
9D1E:          INITADR    .EQ $9D1E  INIT's entry in DOS cmd jump table
AEBE:          FM.INIT    .EQ $AEBE  File Manager's INIT instruction area
BEAF:          RWTS.INIT  .EQ $BEAF  RWTS's INIT instruction area
FDED:          COUT       .EQ $FDED  Monitor's print-character subroutine

BB00:          MOD.GARB       .EQ $BB00  ($E484)  Addresses of FP.GARB
BB08:          MOD.CONTINUE   .EQ $BB08  ($E48C)    entry points after
BBCE:          MOD.CHECK.BUMP .EQ $BBCE  ($E552)    relocation in DOS
BBE4:          MOD.MOVEUP     .EQ $BBE4  ($E568)    nibble buffer.

BB21:          MOD.RELO       .EQ $BB21  ($E4A5)  Points where MOD.GARB
BB52:          MOD.PTCH.2     .EQ $BB52  ($E4D6)    must be changed due to
BBBE:          MOD.PTCH.1     .EQ $BBBE  ($E542)    relocation.

E484:          FP.GARB        .EQ $E484  Address of original garbageman

          *---------------------------------------
          * INCINERATOR's string address buffers
          *---------------------------------------

0022:          NSTR  .EQ 34   # of strings collected on each pass (34 max)

AEBE:          DSC.ADR.LO .EQ FM.INIT          FM.INIT area (AEBE-AF07)
AEB0:          DSC.ADR.HI .EQ DSC.ADR.LO+NSTR  is used to save string
AED2:          DSC.LEN    .EQ DSC.ADR.HI+NSTR  variable table pointers.

BC11:          STR.ADR.LO .EQ $BC11            Left-over part of nibble buf
BC33:          STR.ADR.HI .EQ STR.ADR.LO+NSTR  is used for string pointers;
               *                               ends at $BC56, thus 34 max

          *---------------------------------------
          * INCINERATOR's installation routines
          *---------------------------------------

               CHECK.DOS
4000:18           CLC                     First, check to make sure DOS
4001:A9 00        LDA #$00                  hasn't already been modified.
4003:A2 00        LDX #$00                  Installing the INCINERATOR
4005:A0 01        LDY #$01                  over an existing DOS patch
4007:79 D1 03  .1 ADC P3.DOSWARM+1,Y        could damage disks.
400A:90 01        BCC.2
400C:E8           INX                     Check is made by simply adding
400D:79 1E 9D  .2 ADC INITADR,Y             together the first two bytes
4010:90 01        BCC.3                     in each area to be modified
4012:E8           INX                       and comparing the result with
4013:79 BE AE  .3 ADC FM.INIT,Y             the standard DOS 3.3 result.
4016:90 01        BCC.4
4018:E8           INX                     If DOS has been moved to the
4019:79 AF BE  .4 ADC RWTS.INIT,Y           language card area, or if
401C:90 01        BCC.5                     program is run under ProDOS,
401E:E8           INX                       inclusion of page-3 warmstart
401F:88        .5 DEY                       vector (see $4007) will cause
4020:10 E5        BPL .1                    check to fail.
4022:C9 F1        CMP #$F1
4024:D0 04        BNE .6
4026:E0 03        CPX #$03
4028:F0 43        BEQ INSTALL          DOS ok, begin installation below.
402A:AE 59 AA  .6 LDX $AA59               Can't execute; DOS modified.
402D:C8        .7 INY                       Save DOS stack pointer while
402E:B9 3D 40     LDA .9,Y                  printing error message.
4031:F0 06        BEQ .8
4033:20 ED FD     JSR COUT                Print message.
4036:4C 2D 40     JMP .7
4039:8E 59 AA  .8 STX $AA59               Restore stack pointer and
403C:60           RTS                       quit.

               .9 .HS 8D
                  .AS -"CAN'T EXECUTE."
                  .HS 8D.87
                  .AS -"ACTIVE DOS MODIFIED OR MOVED."
                  .HS 8D.00
403D:8D C3 C1 CE A7 D4 A0 C5 D8 C5 C3 D5 D4 C5 AE 8D
404D:87 C1 C3 D4 C9 D6 C5 A0 C4 CF D3 A0 CD CF C4 C9
405D:C6 C9 C5 C4 A0 CF D2 A0 CD CF D6 C5 C4 AE 8D 00

               INSTALL
406D:CE 1E 9D     DEC INITADR        Point INIT entry in DOS command table to an
4070:A9 56        LDA #$56             RTS--DOS INIT command now does nothing.
4072:8D 0E BE     STA $BE0E          Disable RWTS 'format' command.

4075:A0 00        LDY #$00
4077:B9 C5 40  .1 LDA TEST.GARB,Y    Move INCINERATOR routines to area normally
407A:99 AF BE     STA RWTS.INIT,Y      used by the RWTS init command.
407D:C8           INY
407E:D0 F7        BNE .1

4080:A0 02        LDY #$02
4082:B9 C2 40  .2 LDA .6,Y           Install hook in DOS. Putting hook at $9F2C
4085:99 2C 9F     STA $9F2C,Y          means INCINERATOR will be called whenever
4088:88           DEY                  a <return> is printed.
4089:10 F7        BPL .2

408B:AE 59 AA     LDX $AA59          Save DOS stack pointer while printing msg.
408E:C8        .3 INY
408F:B9 9E 40     LDA .5,Y           Print successful-installation-message.
4092:F0 06        BEQ .4
4094:20 ED FD     JSR COUT
4097:4C 8E 40     JMP .3
409A:8E 59 AA  .4 STX $AA59          Restore stack pointer and
409D:60           RTS                  return to caller; installation done.

               .5 .HS 8D
                  .AS -"THE INCINERATOR IS NOW INSTALLED."
                  .HS 8D.00
409E:8D D4 C8 C5 A0 C9 CE C3 C9 CE C5 D2 C1 D4 CF D2
40AE:A0 C9 D3 A0 CE CF D7 A0 C9 CE D3 D4 C1 CC CC C5
40BE:C4 AE 8D 00
40C2:4C AF BE  .6 JMP RWTS.INIT      Hook to our routines; moved to $9F2C

          *------------------------------------------------------------
          * INCINERATOR main body
          *   These routines are moved into the RWTS INIT space and
          *   are called by DOS every time something is printed.
          *------------------------------------------------------------

               TEST.GARB
40C5:A9 04        LDA #$04             Main entry point. First check to see if
40C7:F0 0A        BEQ INCINERATE         garbage needs to be collected. If
40C9:18           CLC                    lots of space is left, don't proceed.
40CA:65 6E        ADC STREND+1         The $04 at $40C6 means collection
40CC:C5 70        CMP FRETOP+1           occurs only if there is less than
40CE:B0 03        BCS INCINERATE         1K free. Change it if necessary.
               DONE
40D0:4C A4 9F     JMP $9FA4            Return to DOS to continue output.
```

```
          *----------------------------------------------------------
          * INCINERATE
          *   These routines move a copy of Applesoft's original
          *   garbage collector into the DOS nibble buffers, patch it
          *   so that it calls new routines at two critical places,
          *   and then jumps to the new version.
          *----------------------------------------------------------

             INCINERATE
40D3:A0 13      LDY #$13              FP.GARB is Applesoft's original garbage
40D5:B9 83 E5 .1 LDA FP.GARB+$FF,Y      collector. First we move it into the
40D8:99 FF BB   STA MOD.GARB+$FF,Y    DOS "nibble buffers" at $BB00-BC55.
40DB:88         DEY                   These are overwritten on all DOS
40DC:D0 F7      BNE .1                calls, but we can use them for a
40DE:B9 84 E4 .2 LDA FP.GARB,Y          moment here.
40E1:99 00 BB   STA MOD.GARB,Y        MOD.GARB is what we call the new
40E4:C8         INY                   location, since the new copy will be
40E5:D0 F7      BNE .2                patched by the routines below.

40E7:98         TYA
40E8:A0 21      LDY #NSTR-1
40EA:84 46      STY COUNTER           Initialize COUNTER with the number of
40EC:99 33 BC .3 STA STR.ADR.HI,Y       strings (minus 1) and fill the string
40EF:88         DEY                   address buffer with zeros.
40F0:10 FA      BPL .3

40F2:A0 03      LDY #$03
40F4:BE 03 BF .4 LDX RELOCATION.TABLE,Y     Adjust four adrs within MOD.GARB
40F7:B9 07 BF   LDA RELOCATION.TABLE+4,Y     that must change because it's
40FA:9D 21 BB   STA MOD.RELO,X               been moved.
40FD:A9 BB      LDA #$BB                    ($BB is the correct high byte
40FF:9D 22 BB   STA MOD.RELO+1,X             for all relocated adrs.)
4102:88         DEY
4103:10 EF      BPL .4

4105:A0 02      LDY #$02
4107:B9 0B BF .5 LDA PATCH.HOOK,Y      Modify GARB in two places so that it
410A:99 BE BB   STA MOD.PTCH.1,Y        jumps to our new patch routines
410D:B9 0E BF   LDA PATCH.HOOK+3,Y    rather than doing things the old
4110:99 52 BB   STA MOD.PTCH.2,Y      FP way.
4113:88         DEY
4114:10 F1      BPL .5

4116:4C 00 BB   JMP MOD.GARB          Jump to the newly modified garbage
             *                          collector.

BF03:        RELOCATION.TABLE  .EQ *-TEST.GARB+RWTS.INIT
                .HS 00.19.70.EF.9F.95.9F    Data needed for relocation.

4119:00 19 70 EF 9F 95 9F
4120:60         RTS                 Patch for original move-string routine.

BF0B:        PATCH.HOOK   .EQ *-TEST.GARB+RWTS.INIT
4121:4C 60 BF   JMP SAVE.ADRS
4124:4C 11 BF   JMP MOVE.STRINGS

          *----------------------------------------------------------
          *  MOVE.STRINGS is called by our modified garbageman
          *    after a complete scan of the string variable
          *    tables has been completed.
          *----------------------------------------------------------

BF11:        MOVE.STRINGS   .EQ *-TEST.GARB+RWTS.INIT

          *----------------------------------------------------
          *  First find the highest address in our buffer.
          *----------------------------------------------------

4127:A9 22      LDA #NSTR           Initialize counter with the number of
4129:85 46      STA COUNTER            strings our buffers will hold.
              FIND.HIGH
412B:A2 21      LDX #NSTR-1         X is first adr; becomes the highest found.
412D:A0 20      LDY #NSTR-2         Y points to the adr under study.
              HI.CHECK
412F:B9 33 BC   LDA STR.ADR.HI,Y      Compare current to highest found so far.
4132:DD 33 BC   CMP STR.ADR.HI,X
4135:90 0C      BCC NXT.HI            Current is less.
4137:D0 08      BNE NEW.HI            Current is greater.
4139:B9 11 BC   LDA STR.ADR.LO,Y        Can't tell, test low byte.
413C:DD 11 BC   CMP STR.ADR.LO,X
413F:90 02      BCC NXT.HI            Current is less, next please.
4141:98      NEW.HI TYA                    Current is greater, move current
4142:AA         TAX                           pointer to X.
4143:88      NXT.HI DEY               Continue until entire buffer has been
4144:10 E9      BPL HI.CHECK            scanned; X then points to highest
             *                          string in buffer.
```

```
          *----------------------------------------------------
          * Pass this address to a section of the
          *   original garbage collector for moving.
          *----------------------------------------------------

4146:BD 33 BC   LDA STR.ADR.HI,X    Get highest adr
4149:F0 D5      BEQ DONE            If zero, we've moved them all.
             *                        Collection complete, return to DOS
414B:85 9C      STA LOWTR+1         Pass it to MOD.GARB
414D:BD 11 BC   LDA STR.ADR.LO,X
4150:85 9B      STA LOWTR

4152:A9 00      LDA #$00            Remove this address from buffer
4154:9D 33 BC   STA STR.ADR.HI,X

4157:BD B0 AE   LDA DSC.ADR.HI,X    Pass the descriptor address and length
415A:85 8B      STA FNCNAM+1          to MOD.GARB
415C:BD 8E AE   LDA DSC.ADR.LO,X
415F:85 8A      STA FNCNAM
4161:BD D2 AE   LDA DSC.LEN,X
4164:20 E4 BB   JSR MOD.MOVEUP      Tell MOD.GARB to move it.

          *----------------------------------------------------
          *   Repeat until everything in buffer has been moved.
          *----------------------------------------------------

4167:86 6F      STX FRETOP          Adjust FRETOP
4169:85 70      STA FRETOP+1
416B:C6 46      DEC COUNTER         Continue until all adrs
416D:D0 BC      BNE FIND.HIGH         in buffer have been moved.

416F:A0 21      LDY #NSTR-1         Reset counter for SAVE.ADRS
4171:84 46      STY COUNTER
4173:4C 08 BB   JMP MOD.CONTINUE    And continue search.

          *---------------------------------------------------------
          *  SAVE.ADR is called when MOD.GARB has found a string higher
          *    than the lowest string found so far. SAVE.ADR saves
          *    the address of the string and its variable-table
          *    descriptor address and length in our special buffers.
          *  The new string's addresses overwrite those of the lowest
          *    string found up till now.
          *  Afterwards, a scan is made of the buffers to redetermine
          *    the address of the lowest string found so far.
          *    The address of that string is passed to MOD.GARB
          *    and the pointer to it is saved so that any higher
          *    string found will overwrite it.
          *---------------------------------------------------------

BF60:        SAVE.ADRS  .EQ *-TEST.GARB+RWTS.INIT
4176:A4 46      LDY COUNTER          Get current index to adr-save buffers.
4178:99 33 BC   STA STR.ADR.HI,Y     High byte of string adr in A; store it.
417B:8A         TXA                  Low byte of string adr in X; store it.
417C:99 11 BC   STA STR.ADR.LO,Y
417F:A5 5F      LDA INDEX+1          High byte of descriptor address was in
4181:99 B0 AE   STA DSC.ADR.HI,Y       INDEX+1; store it.
4184:A5 5E      LDA INDEX            Low byte of descriptor address was in
4186:99 8E AE   STA DSC.ADR.LO,Y       INDEX; store it.
4189:A5 8F      LDA DSCLEN           Descriptor length;
418B:99 D2 AE   STA DSC.LEN,Y          store it.

418E:A2 21      LDX #NSTR-1          X starts as ptr to first adr in buf;
             *                         becomes ptr to lowest adr in buf.
4190:A0 20      LDY #NSTR-2          Y is pointer to adr now under study.
             LOW.CHK
4192:BD 33 BC   LDA STR.ADR.HI,X     Compare current low to adr under study.
4195:F0 1E      BEQ LDONE               If current low is zero, exit.
4197:D9 33 BC   CMP STR.ADR.HI,Y
419A:90 0C      BCC NXT.LOW          Current low is lower, next please
419C:D0 08      BNE NEW.LOW          New adr is lower, point to it with X
419E:BD 11 BC   LDA STR.ADR.LO,X     Can't tell--test low byte.
41A1:D9 11 BC   CMP STR.ADR.LO,Y
41A4:90 02      BCC NXT.LOW             next please
             NEW.LOW
41A6:98         TYA                  Move pointer aimed at current adr to
41A7:AA         TAX                     pointer for lowest adr
             NXT.LOW
41A8:88         DEY                  Check next adr in buffer till
41A9:10 E7      BPL LOW.CHK             there aren't any more...

41AB:BD 11 BC   LDA STR.ADR.LO,X     Then take the lowest address we have in
41AE:85 9B      STA LOWTR              the buffer so far and put it in LOWTR,
41B0:BD 33 BC   LDA STR.ADR.HI,X       so MOD.GARB will call us if it finds
41B3:85 9C      STA LOWTR+1            any higher strings.

             LDONE
41B5:86 46      STX COUNTER          Save ptr to lowest string as new index.
41B7:4C CE BB   JMP MOD.CHECK.BUMP   Return to MOD.GARB to continue search.
```

A Song Continued

We descended into the murky depths of the Apple Monitor last month, examined memory as a dump of hexadecimal values, as machine language code, and as ASCII characters. I explained how to change the contents of any memory byte from one hex value to another. While these are the most common uses of the Monitor, it can also do other useful and interesting things.

The Monitor was actually designed as an assembly-language-program debugging aid. Nowadays—nearly a decade after Wozniak designed it—much better debuggers are available. Nonetheless, the fact that this one has been built into every Apple ever manufacturered makes its features worth further investigation.

It is difficult to stamp out all the bugs in assembly language programs without some way to stop the program at trouble spots and see what's happening. The standard method for doing this is to use the *break* instruction. The machine language code for a break is $00. To see what happens when this instruction gets executed, let's put one at address $300 and jump to it. Here's how:

```
*300:0

*                                  (press return to check)

0300- 00 02 00 00 00 00 00 00
*300G
                                   (beep)
0302-    A=00 X=00 Y=00 P=30 S=F0
*
```

When you enter *300G*, your Apple executes the break instruction we just put at that address. This causes a "crash" back into the Monitor. You will hear a beep and see a one line display that begins with an address. The address is always two bytes *beyond* the location of the crash-inducing break instruction. The rest of the line shows you the status of your microprocessor when the break occurred. The 6502 microprocessor found in Apples has five special memory locations, called *registers*, that do all the work. These are called the A, X, Y, P, and S registers. The display shows what values were in these registers when the break occurred.

After a break, you can List your program or examine memory. If you need to refer back to what the contents of the registers were when the break occurred, you can do so with the control-E(xamine) command, like this:

```
*                             (press control-E, return)

 A=00 X=00 Y=00 P=30 S=F0
*
```

You can also load the registers with any desired values before executing a routine by using control-E, entering a colon and the desired memory values, and Going to the desired routine. In the following example, we put new numbers in the A, X, and Y registers and jump to our break instruction to see if they're really there:

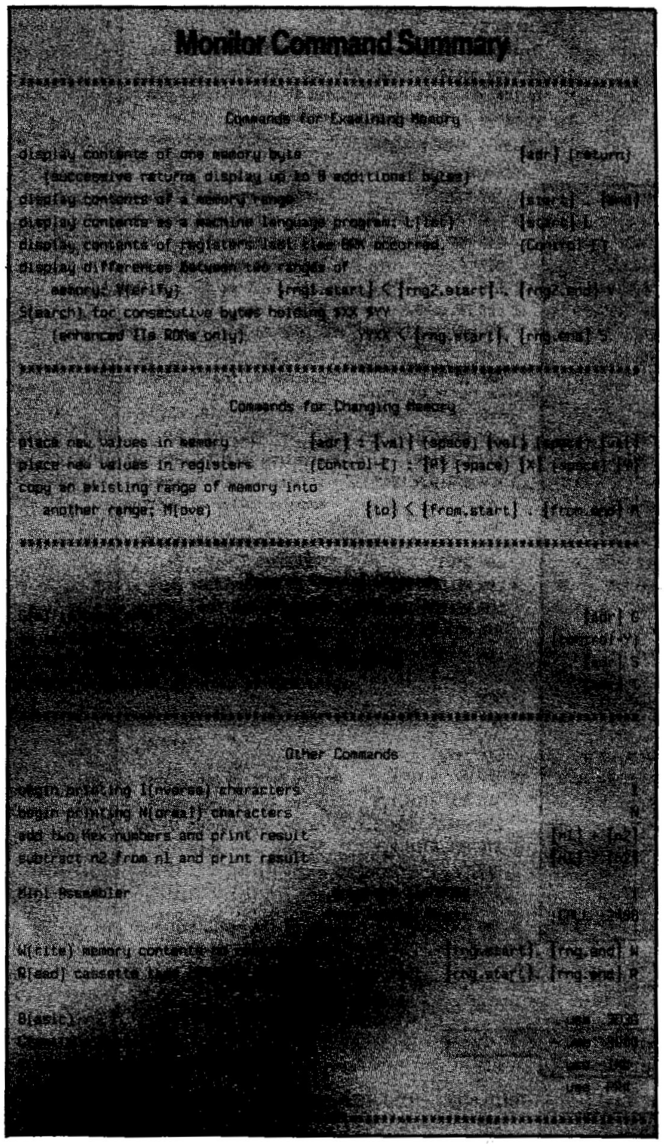
```
*                             (press control-E, return)

 A=00 X=00 Y=00 P=30 S=F0
*:11 22 33

*300G
302-    A=11 X=22 Y=33 P=30 S=F0
*
```

Note: unless you know what you're doing, don't try to change the contents of the P or S registers. The effect can be dramatic.

Meet Mini. The Monitor program built into the original Integer Basic Apple II also had commands for stepping through and tracing machine language programs, as well as a Mini-Assembler. The Mini-Assembler allows you to enter a program in assembly language. It still exists in the Integer Basic image on DOS 3.3 System Master disks, but the S(tep) and T(race) commands are unavailable to all but a few old-timers.

The Mini-Assembler is useful for entering short and quick assembly language programs. It doesn't have many of the features of a full-blown assembler, however (such as labels, equates, and assembler directives), so its usefulness is limited. To get it running, boot a DOS 3.3 System Master disk and enter the *int* command to get into Integer Basic. Then call -2458 (or drop into the Monitor and enter *F666G*).

The Mini-Assembler prompt is the exclamation point. All Monitor commands work within the Mini-Assembler if you start a line with a dollar sign. To begin an assembly, enter the address at which you want the machine langage to go, a colon, and a mnemonic:



Monitor Command Summary

Commands for Examining Memory	
display contents of one memory byte	{adr} {return}
(successive returns display up to 8 additional bytes)	
display contents of a memory range	{start} . {end}
display contents as a machine language program: L(ist)	{start} L
display contents of registers last time BRK occurred	{Control-E}
display differences between two ranges of memory: V(erify)	{rng1.start} < {rng2.start} . {rng2.end} V
S(earch) for consecutive bytes holding $XX $YY (enhanced IIe ROMs only)	YYXX < {rng.start} . {rng.end} S
Commands for Changing Memory	
place new values in memory	{adr} : {val} {space} {val} {space} {val}
place new values in registers	{Control-E} : {A} {space} {X} {space} {Y}
copy an existing range of memory into another range: M(ove)	{to} < {from.start} . {from.end} M
[illegible]	
[illegible]	{adr} G
[illegible]	{Control-Y}
[illegible]	{adr} S
[illegible]	T
Other Commands	
begin printing I(nverse) characters	I
begin printing N(ormal) characters	N
add two hex numbers and print result	{n1} + {n2}
subtract n2 from n1 and print result	{n1} - {n2}
Mini-Assembler	!
	CALL -2458
W(rite) memory contents [illegible]	{rng.start} . {rng.end} W
R(ead) cassette [illegible]	{rng.start} . {rng.end} R
B(asic)	use 3D0G
C[illegible]	use [illegible]
	use [illegible]
	use [illegible]

```
>CALL -2458                        (beep)
!$300.30F                          ($ means this is a Monitor command)

0300- 00 02 00 00 00 00 00 00
0308- 00 00 00 00 00 00 00 00

!300:LDA #CF                       (start adr, colon, instruction)
0300-   A9 CF       LDA   #$CF

! JSR FDED                         (note space before JSR)
0303-   20 ED FD    JSR   $FDED
```

To get out of the Mini-Assembler, press reset.

Enhanced ROM enhancements. The Mini-Assembler can also be found in the new enhanced ROMs for the Apple IIe. You can start up this version of the Mini-Assembler directly from the Monitor by entering an exclamation point.

The enhanced IIe ROMs also have a new *search* command that finds all occurrences of a one- or two- byte value in a specified memory range. The command for finding two sequential bytes with the values $ED and $FD in the range from $F800 to $FFFF is *FDED<F800.FFFFS*. Note that the value to be searched for is entered backward (which makes forward if you're looking for an address pointer that's *stored* backward) and that the line ends with an "S". You can also search for a single byte (*ED<F800.FFFFS*), but you can't search for a two-byte pair if the first byte is a zero.

The IIc's Monitor, although otherwise similar to the Monitor in the enhanced IIe ROMS, doesn't have these features. On the other hand, the IIc Monitor's *list* command (examined last month) correctly disassembles the additional commands in the 65C02 microprocessor's instruction set; this is something neither IIe Monitor can do.

More Commands. There are several fairly useless commands available from within the Monitor. W(rite) and R(ead) are used to save and recall ranges of memory to and from cassette tape. The IIc is missing these commands since it doesn't have a cassette interface.

Still around in all versions of the Monitor, but of little value since the introduction of DOS, are the control-B(asic), control-C(ontinue Basic), control-P(rinter), and control-K(eyboard) commands. Control-B will coldstart Basic, jumping out of the Monitor and erasing any Basic program in memory. Control-C also jumps from the Monitor to Basic, but does a warmstart and leaves any Basic program intact. To keep DOS active, 3D3G (DOS/Basic coldstart) and 3D0G (DOS/Basic warmstart) should be used instead.

Control-P and Control-K are the equivalent to the DOS *pr#* and *in#* commands. Avoid trouble and use the DOS versions.

Two more infrequently used Monitor commands are I(nverse) and N(ormal). These commands affect the way characters are printed. They work just like their Applesoft brethren. N(ormal) is also often used to separate several commands on a single command line, much like the colon is used in Applesoft. We used it that way ourselves last month in line 500 of the Lam Technique demonstration program (page 13). And we use it that way again momentarily.

Monitor magic, with bells. Hexadecimal arithmethic is easy with the Monitor—if you are willing to limit your questions to two digits and can put up with two-digit answers. Watch this:

```
*2F+80
=AF
*80-2F
=51
*80-90
=F0
*80+90
=10
```

Only addition and subtraction work. Multiplication and division are not supported.

The Monitor includes commands that allow you to move a range of memory to a new location or to compare two ranges of memory with each other and print the differences. The format for these commands is {*destination*} < {*start*}.{*end*} followed by either a M(ove) or V(erify) command character. Note how the less-than sign forms a little arrow that graphically displays which way the bytes will be moved. Here's how you use them:

```
*800<3D0.3FFM         (moves the $3D0-$3FF range to $800-$82F.)

*800<3D0.3FFV         (verify that the move took place--no response
                          means that it did.)

*9E51<3D0.3FFV        (compare the $3D0-$3FF range to $9E51-$9E80.)
03F2-BF (4C)            (discrepancies:
03F3-9D (65)              number after dash is value at address shown
03F4-38 (FF)              number is parantheses is value at corresponding
                            address in comparison range.)
*
```

The first line above demonstrates the M(ove) command. This command copies an image of the $3D0-$3FF memory range at $800-($82F). The image at $3D0 is left unchanged, although this is not always the case, as we'll see shortly.

The second line demonstrates the V(erify) command. Here we ask the Monitor to compare the $3D0-$3FF memory range with what's at $800. It does so and reports all discrepancies. Since our move command just made these ranges match, there are no discrepancies to report.

The third line compares the $3D0-$3FF range with a range of memory inside DOS 3.3 at $9E51-$9E80. The two areas should be similiar, since Uncle DOS moves what's at $9E51 to $3D0 whenever a disk is booted. As you can see, three bytes are different. The number after the hypen shows the value at the address displayed on the screen; the number in the parentheses shows the corresponding value in the range being verified.

Now, consider what happens when the range of memory we are moving *overlaps* the range of memory we are moving to. If we are moving a memory image *lower*, the move will take place normally. In the process, however, the overlapped part of the original image will be destroyed.

When we move a memory image higher, on the other hand, an interesting thing happens. The portion of the image that does not overlap is repeated throughout the new range. You can quickly fill a range of memory with a specific value or pattern of values, for example, like this:

```
*2000:FF N 2001<2000.200EM  (Put FF at 2000, use N(ormal) as a command
                                separator, M(ove) contents of 2000-200E to 2001.)
*2000.200F   (Display new contents of 2000-200F.)
2000- FF FF FF FF FF FF FF FF
2008- FF FF FF FF FF FF FF FF
*
```

The *2000:FF* initializes the first byte of the range. The *N* tells the Monitor we are done with the memory change command and now want to enter another command. The *2001<2000.200EM* moves the value now at byte 2000 into bytes 2001 through 200F.

The generalized format for the M(ove) command used in this way—when *length* indicates the length of the pattern to be repeated—is:

```
{start+length} < {start} . {end-length} M
```

The machine language command 20 DD FB will beep your Apple's speaker. Want to hear a thousand beeps? Try this:

```
2000:20 DD FB N 2003<2000.2BB4M N 2BB8:60 N 2000G
```

That command line only takes about 114 seconds to execute. The *2BB8:60* puts a machine language command at the end that returns us smoothly to the Monitor. Try *2000L*. Then try *LLLLLLLL*. Amazing stuff.

Or perhaps you'd prefer:

```
*N FBDDG 34:0        (put a space after the zero, before you press return)
```

This trick creates a repeating command. Start the line with a single letter command, end it with *34:0(space)*. This pokes a zero into byte $34, which causes the Monitor to start executing the line over again at the beginning. If you want to start beyond the beginning, replace the zero with the number of the character you want to start with, minus one.

Enough tricks. The point is that a familiarity with the Monitor, *which is available in every Apple II ever built* (and few other kinds of computers), gives you the power to control your machine at a very elementary level. The box on the previous page summarizes the Monitor's commands.

Help for the absent-minded

Absent-mindedness is a common fault, especially around here. Consequently, we have a great deal of sympathy for those who manage to destroy important files by mistake.

Word processing and spreadsheet files are particularly vulnerable. The problem is that most programs' *save* and *load* commands are easily mixed up. A frequent cause of file loss is that a user absent-mindedly saves a blank screen over the top of a valuable file. Listen carefully on any clear evening and you will hear crys of file-loss anguish emanating from chimneys in most major cities.

However, if the word processor or spreadsheet program uses DOS 3.3 text files, the lost data is easily recoverable. This is because DOS 3.3 doesn't remove the old file from the disk. Instead the new contents are written over the top of the old contents and an end-of-file mark (control-@, hex $00) is placed at the end of the new stuff. If the new contents consist of nothing (a blank screen), only one character of the original file is destroyed (the one under the new end-of-file mark). To recover the file, all you have to do is replace that file-ending zero with something else.

Typically this is done using a disk-edit utility. I once used this technique to recover a large section of a friend's Masters Thesis and prevented a potential suicide. The technique requires some expertise, however.

A better way, once you get DOS 3.3's *append* command working (see the back page of this letter) is to let *append* find the end-of-file mark, then overwrite it with a carriage return. When the file is reloaded into the word processor or spreadsheet, it will burst forth virtually unscathed (damage to word processor files will be at the beginning; damage to spreadsheet files will be at the *end*). Here's a program that uses this trick. Go out and save a few lives with it.

```
10 REM     *** Phoenix ***

100 HOME : VTAB 12 : D$=CHR$(4)
110 PRINT "PLEASE INSERT DISK WITH LOST FILE."
120 PRINT "  WHAT IS THE NAME OF THE FILE?"
130 PRINT
140 INPUT "";F$

150 PRINT D$;"UNLOCK";F$ : REM Create error if file-not-found.
160 PRINT D$;"APPEND";F$ : REM Move file pointer to end-of-file marker,
170 PRINT D$;"WRITE";F$  : REM   overwrite marker with a <return>.
175 PRINT
180 PRINT D$;"CLOSE";F$  : REM Rest of file magically reappears.

190 PRINT "FILE FIXED!" : END
```

Ask (or tell) Uncle DOS

Loose Ends

Last month I left some questions unanswered, and a couple of them still have me puzzled. But readers have helped with answers to some.

*Regarding **AppleWorks** and non-Apple printers and interface cards: Apple has released an updated version of **AppleWorks** to dealers that solves this problem. Dealers are supposed to update your original disks for free. Call ahead to make sure your dealer has the update. My source on this one highly recommends you insist on talking to the dealer's technicians, rather than salespeople, for this and all similar problems with Apple programs and equipment. It's the technicians that Apple notifies about such bugs and how to fix them. My source adds that the ProDOS version of **AppleWriter** has the same problem—dealers have also been issued a fix for this.*

One of last month's letters asked how to do an 80-column screen dump. In the response to that letter, I mentioned that my printer garbled when I printed while the 80-column screen's even columns, which are stored in auxiliary memory, were turned on. But I didn't know why. The reason occurred to me as soon as the letters were printed. Printer interface cards, like most Apple II peripherals, use a small amount of memory inside the text screen. This area is usually called the scratchpad area, *or the* screen holes. *When the even columns are on, the scratchpad area the interface card is using disappears; thus the problems.*

If you need a faster 80-column screen dump, delete line 305 from last month's program (page 15), remove the "L$+" in line 340, change line 360 to "PRINT L$+CHR$(PEEK(ADR));", and remove the "L$" from line 380.

Remittance bug revealed

Tom, try this program:

```
10 Y1=24 : REM 1 year subscription price
20 Y2=44 : REM 2 year subscription price
30 PRINT (Y1*2)-Y2
RUN
4
```

Why do your bills say a two-year subscription is an $8 savings?

Philip Straus
Philadelphia, Pa.

My error. Abby, ***Open-Apple's*** *spokeswoman, nearly resigned when she saw your letter. She agreed to continue with* ***Open-Apple*** *only if I tacked a couple of extra months onto the subscriptions of those of you who actually paid $44, and I've done so.*

Input comma no garbage

I have an addition to the Selective String Preservation technique for avoiding garbage collection that you wrote about in the January issue (pages 4-5). Here's a routine that replaces Applesoft's *input* command so that commas are allowed. Normally routines that use *get* like this cause garbage collection to occur frequently, but this one doesn't create any more garbage than *input* itself does. As written, the routine also filters out control characters.

```
50 SL=PEEK(111) : SH=PEEK(112)
51 I$=""
52 FOR C=1 TO 255
53 : GET A$ : PRINT A$;
54 : IF ASC(A$) = 13 THEN C=255 : GOTO 56
55 : IF ASC(A$) > 31 THEN I$=I$+A$
56 NEXT
57 PRINT
58 POKE 111,SL : POKE 112,SH
59 I$=MID$(I$,1)
```

I$ is made permanent by line 59, which makes a new copy of it after the pointers at 111 and 112 have been changed to erase the temporary copies of A$ and I$. They will be overwritten and won't cause garbage collection.

Jim Parr
Bloomington, Ill.

Fast garbage bugs

I tried the fast garbage collector published in the January 1981 *Call -A.P.P.L.E.* and mentioned in your January issue. I had some trouble getting it to work. After studying the code, I discovered some bugs. The subroutine NZTAB, which zeros out the address table, ends with a JMP to FNDVAR2, not an RTS. Consequently, the initial call to NZTAB should be a JMP, not a JSR, and should occur just before FNDVAR2, between lines 63 and 64, not at line 59.

In addition, the Y register should be cleared when a new address is placed in INDEX. Add a LDY #0 just before lines 69 and 79.

Steve Hunt
Cambridge, Mass.

There was one more bug. Val Golding, editor of Call -A.P.P.L.E. *when the original program was published, tells me he misspelled the author's name and I reprinted his mistake. The author's name is Randy Wigginton. It's an important name to spell right, as Randy has been involved with Apple since, as a 16-year-old in 1976, he hitched rides to Homebrew Computer Club meetings with Steve Wozniak. He's been one of Apple's most prolific wizards ever since.*

Rana, A.P.P.L.E, Abacus

Your February editorial about DOS 3.3 was on the mark, and raises an interesting question. How do people who make double-sided drives (e.g. Rana Elite II) make them compatible with DOS 3.3? Do they define a logical track to be two physical tracks—presumably one on each side of the disk? This would create, in effect, tracks with 32 sectors, which you say DOS 3.3 can handle. I imagine this scheme would not require any repositioning of the head to read all data from any logical track.

Speaking of Rana, while I still see their products being advertised by a number of mail order houses, they appear to have completely discontinued their own advertising. I was greatly interested in their 8086/2 unit since it provides MS-DOS compatibility for the Apple as well as high capacity drives usable under all four operating systems that run on the Apple (DOS 3.3, Apple Pascal, CP/M-80, and ProDOS). I obtained a pre-publication copy of the 8086/2 manual, and actually saw a unit once at a dealer, though like most dealers he couldn't answer a single question. A friend of a friend has one, and while the level of compatibility with the IBM PC is not as high as one would like, he reportedly is satisfied with it. I wrote a lengthy letter to Rana asking a number of questions about six months ago and never got an answer. I'm still interested. I still have all those unanswered questions and one more besides: How come you don't hear anything about a product that once seemed destined to sell several hundred thousand copies at over $1,500 each?

I would like to call attention to a situation that I feel is deplorable, especially since it involves a group in which I had placed considerable trust and which I had enthusiastically recommended to many friends. The billing policy of A.P.P.L.E. (the user group in the Seattle area that publishes *Call -A.P.P.L.E.* magazine) doesn't conform to the standard practices of ethical mail order merchandising. On December 1, I ordered several software packages from them by mail. They billed my credit card on December 11, and as of late January they had not yet shipped me anything and refused to make a commitment on when they would ship. Standard practice is to bill only at the time shipment occurs. I like to think of A.P.P.L.E. as a group of the Apple community's most upstanding citizens, so this incident comes as a grave disappointment.

On a more pleasant note, I would like to recommend the 128K RAM card manufactured by Abacus Enterprises, P.O. Box 1836, Detroit, MI 48231 (313-524-2444 voice, 313-524-0238 300 baud). I think it is a super product and the service provided by Abacus is outstanding. The hardware is unique in that an external switch allows the card to emulate either the Saturn or Legend bank switching protocols. The card can be write protected, also by means of an external switch. Abacus has released a software package called Back-to-Back that allows swapping images of the lower 48K of RAM into and out of the 128K card at the press of a button. This makes it possible to suspend operaton of one program, toggle to a second and, at the press of a button, resume operation of the first right where you left off. Abacus also provides a very easy-to-use RAM disk emulator for use with DOS 3.3.

Dan Strassberg
Arlington, MA

Rana provides a highly modified version of DOS 3.3 with its high capacity drives. It actually has two Volume Table Of Contents sectors on each disk and is consequently incompatible with lots of stuff, including **ProntoDOS**. *I picture Rana as a company that tries just a little bit too hard. Their products do wonderful things nobody else can do, but at the price of compatiblity problems that keep them out of the fast lane.*

Like you, I've bought tons of stuff from A.P.P.L.E. over the years and have recommended them in this letter and other places. Like you, I recently had an order delayed. The folks in Seattle say they ran into unexpected problems with a cross-town move they made in late December, and that is why customers are experiencing delays.

As a certified mail order junkie, I commend you for having the good sense to pay by credit card, and I recommend that everyone buy mail order goods that way when possible. If A.P.P.L.E. still hasn't shipped, complain to your credit card company and let them handle it. That will get A.P.P.L.E.'s attention.

A charming difference

You say the S.H. Lam technique for entering Monitor commands from inside Basic programs (Feb pages 12-13) works from within a subroutine. But the same routine appears on the Beagle Bros **Peeks, Pokes and Pointers** chart and the chart says it *won't* work within a subroutine. Who's right?

Uncle Louie
San Diego, Calif.

Both of us. In the version shown on the Beagle Bros chart, ***D823G*** *is tacked onto the end of the Monitor command string. In the* ***Open-Apple*** *version,* ***D9C6G*** *is tacked on. Both formulae cause the Monitor to jump to charmed spots within Applesoft, however, one spot is more charmed than the other and works from within subroutines.*

Who needs *OPEN*

In January's *Digging Into DOS* (page 2), the straightforward way you give to read a text file contains an extra step. For reading text files, *open* is a redundant command. If you want to *open abc* and read from it, just *print d$;"read abc".*

Looking at the code for read, it's easy to see why *open* is not required. The first thing DOS does in *read* is to check whether the file is open. If not, it opens it. I no longer use the *open* command for reading text files. An added advantage is that a file is not created if it doesn't already exist—you'll get a *file not found* error instead.

The *write* command works in a simliar manner.

Charles H. Putney
Shankill, Co. Dublin, Ireland

You are absolutely right. The only problem with this trick is that Apple never documented it; consequently the ProDOS developers apparently didn't know about it; consequently it doesn't work with ProDOS.

Disk free space

In the March 1984 issue of *Softalk*, you wrote about the DOS 3.3 Volume Table of Contents and you printed a Basic program to calculate the number of free sectors on a disk. I have used that program as a subroutine in a larger program, which is now running under your *ProntoDOS*. I am getting very limited by having DOS at 48K, and the program is causing Applesoft's garbage collection to occur very frequently. I would like to use your *DOS-UP* program on the *ProntoDOS* disk to move DOS up to the top 16K. Can you tell me where DOS's VTOC buffer is located after DOS has been moved?

Robert Nyberg
Tucson, Ariz.

It's 16,384 ($4000) bytes higher than before, but that information won't help you much. If you look at that address range from within a Basic program, what you'll see is the machine language code of Applesoft itself, not DOS. If you turn the upper 16K of memory on so you can see DOS, your Basic program will crash because Applesoft will disappear.

You have to peer at the high reaches of memory with an assembly language routine. A better approach to solving your problems may be to use the garbage collection tips published in this issue and in January's.

Although I've written a program for moving DOS to the upper 16K of memory and although many, many people use it, my experience has been that moving DOS often causes as many problems as it solves. If you are writing pure Basic with no assembly language routines and no fancy tricks, a relocated DOS works great. But as soon as you try to fine tune things, problems begin to occur.

Another approach to consider is to switch to ProDOS. This won't increase the amount of memory available, but it gives you fast garbage collection and a chain command that's quick and easy to use. On 128K machines you'll have the ProDOS /RAM disk; you can use it to keep the chained parts of your program available for quick access. Finding the number of free blocks on ProDOS disks is pretty easy. Fish it out of the catalog, which you can open and read like any other file.

80-columns & machine language

How do you activate or deactivate Apple's 80-column card from within a machine language program? After the card is activated, what routine is used to write a character to the screen?

Tom Carlin
Cleveland, Ohio

First, the easy stuff. Once you have the card turned on, you send characters to it from machine language by loading the character into the A register and doing a JSR to COUT ($FDED)—exactly the same procedure as used in 40 columns. To deactivate the card, simply use this procedure to print a Control-U.

Other fancy things you can do by sending control characters to COUT with 80-column mode activated are:

control chr	ASCII val	name	action
cursor moving codes/none clear any part of the screen			
H	$08	backspace	left one character
\	$1C	fwd. space	right one character
J	$0A	line feed	down one line
M	$0D	return	left edge and down one
Y	$19	home	upper left corner
text moving codes/neither moves the cursor			
W	$17	scroll up	text moves up
V	$16	scroll down	text moves down
screen clearing codes			
L	$0C	clear	whole screen
K	$0B	clear EOS	cursor to end of screen
]	$1D	clear EOL	cursor to end of line
Z	$1A	clear line	clear line cursor is on
other codes			
O	$0F	inverse	begin inverse display
N	$0E	normal	begin normal display
Q	$11	40-column	begin 40-column display
R	$12	80-column	begin 80-column display
U	$15	quit	turn 80-column card off

Turning the card on varies depending on whether you want DOS to remain connected or not. If you are using DOS 3.3 and you leave it connected, Uncle DOS will respond to commands that are preceeded by a return and a control-D, just as he does from Basic. Simply send the command to COUT one letter at a time. In this situation, you turn on the 80-column card by sending DOS the command PR#3.

ProDOS does not respond to commands sent to COUT from assembly language programs. You must use the ProDOS machine language interface, or, if your machine language program is CALLed from a Basic program, you can poke the command string into the keyboard input buffer at $200 (don't forget to put a return at the end), and do a JSR DOSCMD ($BE03) to execute it. Once again, use PR#3 to turn on the 80-column card. (CAUTION: several commands, including read, write, *and* append, *don't work correctly when initiated with DOSCMD.)*

If you don't need DOS, turn on the card like this:

```
A9 03       LDA #$03      80-column slot number
20 95 FE    JSR OUTPORT   do a pr# with $FE95
A9 8D       LDA #$8D      return
20 ED FD    JSR COUT      print it with $FDED
```

Big Boy Data Managers

Is there a good database manager for the Apple IIc that will accomodate 15-20 thousand names and cross reference them with about 25 categories?

Lowell Levinger
Inverness, CA

I haven't tried all of the database programs available for the Apple, but of the ones I have used, the database manager in **AppleWorks** *is my run-away favorite. I think it's the strongest of the three programs that comes with* **AppleWorks**. *It won't do what you want, but keep reading.*

After trying for a couple of weeks to set up **Open-Apple's** *subscriber records with a "high-powered" Apple II database manager (and getting nowhere), I turned to* **AppleWorks** *and had the whole thing set up, complete with several kinds of status reports, in a single afternoon.*

Since then I've written some Basic programs that read **AppleWorks***-generated text files and do special manipulations (such as, while printing monthly mailing labels, also printing an invoice label for the remittance envelope if an account is unpaid).* **AppleWorks'** *ability to generate standard text files holding data it has sorted, selected, and formatted is a very strong feature of the program.*

If you have a IIc, you should have **AppleWorks** *anyhow, so if I were you I'd start with it. It's quite easy to use and is a good place to organize and practice using a database. It won't be your final solution, however—the great limitation is that you won't be able to get anywhere near 15,000 records in a single file. Using a IIc you'd probably need about 30 separate files to hold that much data. There is a possibility this wouldn't be a problem for you, but that's unlikely.*

By starting with **AppleWorks**, *however, you'll have a clearer idea of what you want to do and how to do it. Its great advantages are speed, ease of use, and clarity. Sorting a 500 record file takes less than 10 seconds. You can move from record to record instantaneously. You can display a whole screenful of records at the same time and scroll through them.*

Once you see what **AppleWorks** *can do with a sample of your data, you'll be in a better position to judge other database managers by their instruction manuals. Given ProDOS, there is no theoretical reason why a IIc shouldn't be able to handle a database as large as yours. I would recommend a hard disk or other high-capacity storage device to avoid finger blisters caused by constantly opening and closing disk drive doors, however.*

It's also necessary to consider how often the records in your file are to be updated. If the records consist of something like daily meal requests for 15,000 people, the Apple IIc will be woefully inadequate simply because one person working at one computer can't update that many records in one *day. If the records are updated yearly, on the other hand, a single person using good software and a IIc will be able to manage the data and do lots of other stuff, too. Always keep in mind the limits of what one person using one computer can do when organizing this much data. Sometimes you just can't handle it without going to some kind of multi-user system.*

Another big boy

A company here in Texas called Applied Engineering has a card for the Apple IIe that can expand memory to as much as 1 megabyte (1,024K bytes). The company also offers a patch for **AppleWorks** that will allow the user to have an 800K desktop. That should quell some of the blithering about how limited **AppleWorks** is due to memory restraints.

For the price of an IBM PC and Lotus XYZ a smart user could have a IIe with 1 meg of RAM, **AppleWorks**, and the Sider hard disk drive. There might even be enough left over to buy a joystick. This would turn the IIe into a real mother-hummer. It might also make a few of the PC parrots choke on their crackers.

Gary Maddox
Weatherford, Texas

You think just like I do. I have one of Applied Engineering's cards on order—look for a complete report here in a month or two.

The DOS 3.3 *append* challange

Could you discuss the bug in the *append* command of DOS 3.3? When the file being appended to ends at the end of a sector, it doesn't work. I have discussed this with Apple, but they had no help to offer. Is there a way to fix this?

Ferd G. Fender
Glenview, Ill.

DOS 3.3's append *command graphically demonstrates the difficulty of getting all the little bugs out of complex software. Apple has officially modified* append *three times and still doesn't have it right. I have officially modified it once and didn't get it right either. So it's with some fear and trembling that I enter this discussion.*

Append *is very similar to the DOS* open *command. You use it to get a specific text file ready for reading from or writing to. When you use* open*, DOS aims its* position-in-file *pointer at the first byte of the file. When you use* append*, on the other hand, DOS aims its pointer at the byte just beyond the last byte in the file. If your next action is to write to the file, what you write will start at the end of the file, rather than the beginning, and thus will be appended, or added to, the original file. (If your next action is to read from the file, you'll get an* end of data *error.)*

Unlike ProDOS, DOS 3.3 doesn't keep a record of how long text files actually are. When you issue an append*, DOS 3.3 simply starts reading the file and continues until it comes across a byte holding a zero. By definition, the first zero byte encountered marks the end of the file. However, by the time DOS retrieves the zero, the* position-in-file *pointer is aimed at the next byte—the byte beyond the zero. So DOS next executes an internal* position *command to back up the* position-in-file *pointer by one byte.*

So far this all seems pretty simple, right? What I've just described is exactly how the DOS 3.2.1 append *command worked. But there's a bug here. While a zero byte in a file is the* **main** *method for marking the end of the file, it's also possible for a file to simply have no more sectors. This happens whenever a file's length is a multiple of exactly 256 bytes. The final sector in the file will be completely filled with data, there will be no more sectors assigned to the file, and there won't be a zero anywhere in sight. In this situation, the* position-in-file *pointer doesn't get bumped up an extra notch, yet* append *insists on bumping it back a notch as usual. The effect is that the final byte of the original file is overwritten by the appended material whenever a file ends exactly on a sector boundry.*

When the original version of DOS 3.3 was released (around here I call it DOS 3.3.0), a patch was added to take care of the problem. Unfortunately, what should have been a simple test in the append *routine to determine whether the* position-in-file *pointer should be reset or not became a 74 byte patch of amazing complexity. However, it fixed the bug.Yet, like many patches, it added a new bug of its own. This bug is quite obscure—let's just say that when the stars are right, the DOS 3.3.0* append *command will still fail to work.*

It was about this time that yours truly wrote **ProntoDOS** *for Beagle Bros. Much of the room for the* **ProntoDOS** *routines came from removing Apple's 74-byte patch. I solved the major* append *bug and got rid of the exotic one with just a few bytes of code. I bragged that I had solved the DOS 3.3* append *problem. Such fat-headedness is always inappropriate for assembly language programmers.*

For there was yet another bug. The position *routine that DOS calls to notch back the* position-in-file *pointer doesn't work correctly if the file being appended to is longer than 32,767 bytes. Art Schumer described the bug in an article that appeared in the August 1982* Call -A.P.P.L.E.*, page 57. (It's also in* Call -A.P.P.L.E. In Depth #3: All About DOS*, page 191.) Unfortunately, I didn't come across the article until several months after I had finished* **ProntoDOS**.

The folks at Apple saw the article, however, and when they released a new version of DOS 3.3 for the IIe in January 1983 (I call this version DOS 3.3e), they fixed this bug. Rather than using position *to notch back the pointer, the IIe version modifies the pointer by hand. Apple also added 13 additonal bytes of code to fix the more exotic bug in their original 74-byte patch.*

Was everybody happy? Does this story have a pleasant ending? Not yet, gentlepeople; Apple's new patch made things even worse.

The position-in-file *pointer within DOS is three bytes long. One byte is aimed at the byte offset within a sector. The next byte is aimed at the sector number—until it hits 256 anyhow. Then the third byte comes into play and the second byte starts over again at zero.*

Apple's DOS 3.3e patch only adjusts the lowest of these three bytes. When a file-ending zero appears in the final byte of a sector, the middle byte of the pointer will have been advanced to point at the next sector. But since the patch doesn't decrement the middle byte, the position-in-file *pointer ends up aimed 256 bytes beyond where it should be. DOS 3.3e's* append *fails completely once every 256 times.*

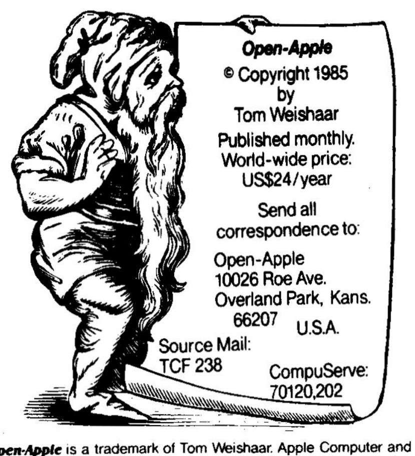
In September 1983 Apple released a third version of DOS 3.3. Apple's people said this would be the final update to DOS 3.3, since ProDOS would be introduced within a few months. (I call this final fairly-fixed version DOS 3.3f). This version expands the 3.3e patch to decrement the middle byte of the position-in-file *pointer correctly. However, it still neglects to decrement the high byte of the pointer when necessary; consequently even this version has an* append *that will fail if you try to add something to a file that's exactly 65,535 bytes long.*

The program that follows fixes all the append *bugs I know about in all three versions of DOS 3.3. Enter and run the program and test the results on a few files. If everything works, you can initialize a new disk and the changes will become a permanent part of the DOS on your new disk. With standard DOS 3.3, the patch overlays a portion of Apple's useless 74-byte* append *patch and thus requires no additional space.*

ProntoDOS*, however, already uses that area for something else. If you are a Pronto user, start with an unmodified copy fresh from the original Beagle Bros disk. Then enter, run, and test this program. After the* append *patch has been added, you can use* Pronto Update *to add most, but not all, of the other* **ProntoDOS** *enhancements and to update your disks. Don't run this program* **after** *making other enhancements to* **ProntoDOS** *or trouble could result.*

(If you are a **DiversiDOS** *user, incidentally, your append command already works correctly.)*

```
10 IF PEEK(978)<>157
     THEN PRINT "48K DOS 3.3 NOT ACTIVE." : END
15 PRINT "Installing APPEND patch..."

20 REM identify DOS type
21 ID=PEEK(46725)
22 IF ID=165 THEN DT$="DOS 3.3.0" : GOTO 30
23 IF ID=186 THEN DT$="DOS 3.3e"  : GOTO 30
24 IF ID=182 THEN DT$="DOS 3.3f"  : GOTO 30
25 IF ID=206 THEN DT$="ProntoDOS" : GOTO 40
26 PRINT "ACTIVE DOS NOT RECOGNIZED." : END

30 C$="A2A1:92 B6" : GOSUB 500
32 C$="A6B3:0A"    : GOSUB 500
34 A$="B692"       : GOTO 50

40 C$="A6C0:B3 B6" : GOSUB 500
44 A$="B6B3"

50 C$=A$+":B0 18 AC E6 B5 D0 10 AC E4 B5
          D0 08 AC E5 B5 F0 0C CE E5 B5
          CE E4 B5 CE E6 B5 20 7E AE 60" : GOSUB 500
52 PRINT DT$;" currently active."
54 PRINT "APPEND now works with any size file."
56 END

500 C$=C$+" N D9C6G"
510 FOR I=1 TO LEN (C$) :
      POKE 511+I, ASC(MID$(C$,I,1))+128 : NEXT
520 POKE 72,0 : CALL -144
530 RETURN
```


Open-Apple
© Copyright 1985
by
Tom Weishaar
Published monthly.
World-wide price:
US$24/year
Send all
correspondence to:
Open-Apple
10026 Roe Ave.
Overland Park, Kans.
66207 U.S.A.
Source Mail:
TCF 238
CompuServe:
70120,202