Open-Apple™

Releasing the power to everyone.

September 1985
Vol. 1, No. 8

Input absolutely anything

Two major problems Applesoft programmers have when trying to write polished software are the unpredictable delays caused by garbage collection and the inability of the INPUT command to accept commas or colons. Apple fixed the garbage collection problem with ProDOS and ***Open-Apple*** has shown you how to control it under DOS 3.3 (January, pages 4-5; March, pages 17-19).

Apple also wanted ProDOS to fix the INPUT command. Appendix B of *Basic Programming with ProDOS* says (page 207), "The Applesoft INPUT command has been made more useful....When you use ProDOS, the last variable in the INPUT list is assigned all the remaining characters in the line, including commas and colons. This means that you can now use a single INPUT statement to read in any arbitrary string of characters." Many people have been very excited about this feature. Unfortunately, it's only in the manual — ProDOS and Basic.system handle INPUT exactly as Applesoft and DOS 3.3 do.

A better keypunch machine. The reason INPUT won't accept commas or colons is that Applesoft was designed so that several variables could be input at one time. Consider this little Applesoft program:

```
100 PRINT "ENTER YOUR NAME, AGE, AND EYE COLOR."
110 INPUT N$,A,E$
120 PRINT "THANKS" : END
```

When you run this program, you can enter all three data elements at one time, separated by commas. Or you can enter them one at a time. If the data is entered correctly (a big assumption), here's what appears on the screen in each situation:

```
ENTER YOUR NAME, AGE, AND EYE COLOR.
?molly,8,bluish-brown
THANKS

ENTER YOUR NAME, AGE, AND EYE COLOR.
?joe
??4
??brownish-blue
THANKS
```

Basic was developed in the 1970s, when trained operators entered most computer data on keypunch machines. This kind of INPUT flexiblity seemed on the cutting edge of technology in those days. Now, however, after programs like *GPLE, VisiCalc,* and *Apple Writer,* INPUT seems absolutely decrepit.

There is no way you can write an Applesoft program that has data-entry and editing facilities as good as those in, say, AppleWorks. So why not use AppleWorks or a similar program for data entry instead of even bothering trying to write your own data-entry routines? If your data is mostly numbers, use a spreadsheet. If it is mostly words, use a word processor. If it is a mix of different types of information that have a consistent format, use a data base. Last month *Open-Apple* showed you how to get these kinds of data out of AppleWorks and into files you can manipulate with Applesoft programs. For tabular, numeric information, use DIF files. For other types, use standard ASCII text files.

The venerable GET loop. So we agree that in 1985 Applesoft's INPUT command shouldn't be used for much besides reading data off of disks. But unfortunately, because of the way its developers fancyfied it back in the 1970s, it doesn't even work very well for that. Consider what happens when you try to read these lines of data from a disk file with INPUT:

```
Route 4, Box 92
Pascagoula, Mississippi
Apple Computer, Inc.
9/15/85 08:30
```

In each case, an EXTRA IGNORED error message will appear on your screen (although program execution will continue uninterrupted) and you will lose all information after the first comma or colon in a line.

An easy way to solve the problem is to use a GET-loop instead of INPUT:

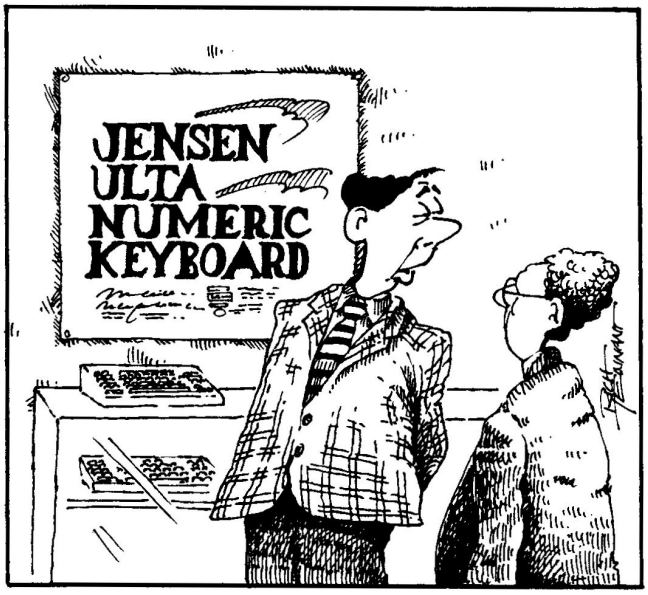
```
100 GET A$ : IF A$<>CHR$(13) THEN B$=B$+A$ : GOTO 100
```

In a GET-loop like this one, A$ is a temporary variable. The completed string ends up in B$.

There are three problems with this technique. First, it is extremely slow. Second, it creates tons of garbage. Third, if you attempt to input a line with more than 255 characters before a carriage return, you will get a STRING TOO LONG error.

(Applesoft strings have a maximum length of 255 characters. The Applesoft INPUT command, however, will accept only 239 characters. This appears to be because INPUT and the section of Applesoft that reads program lines you type on the keyboard share some important subroutines. Program lines are limited to 239 characters; because of the shared subroutines INPUT is too, but it doesn't have to be. Interestingly, Applesoft will allow you to enter 255 characters of both program lines and INPUT, but everything after character 239 is ignored without warning.)

In the March ***Open-Apple*** (page 22), subscriber Jim Parr gave us a nine line Applesoft subroutine that avoids the garbage-creation and string-too-



"FOR OBVIOUS REASONS THEY DECIDED NOT TO USE AN ACRONYM."

long problems. It is suitable for any program that can access files casually. If you want to do some heavy-duty data input, however (like reading a couple thousand subscriber records of 30 fields each from a disk file), a GET-loop quickly adds *minutes* to program execution time.

An historic look at input-anything routines. Because of the problems INPUT and GET-loops have, Applesoft programmers have been looking for a better solution for a long time. The earliest input-anything subroutine I have heard of was published in *Contact*, a programming newsletter Apple itself published in 1978 and 1979. The routine appeared in the October 1979 issue, in the early days of Applesoft. To use it, the first variable you define in a program has to be X$. Then you have to poke in a short machine language program that starts at byte 768. Then, instead of INPUT "ENTER YOUR ANSWER HERE:";A$, you say:

```
500 PRINT "ENTER YOUR ANSWER HERE:";
510 CALL 768 : A$=MID$(X$,1)
```

The assembly language routine at 768 calls the Monitor's NXTCHAR routine, which gets characters from the keyboard or other input device, prints them on the screen as they are typed, stores them in the keyboard input buffer at $200, and stops (returning the length of the string in the X register) when a return character is typed. NXTCHAR supports the standard (and minimal) Monitor editing functions, such as back space and forward space.

After using NXTCHAR to input the string, the routine puts the string's length and the address of the keyboard input buffer into the *variable descriptor* for X$. Applesoft uses variable descriptors to keep track of the current value of each variable. The variable descriptors are kept in a *variable table* Applesoft builds as a program executes. Each descriptor indicates to Applesoft what kind of variable it describes (integer, real, string, function, integer array, real array, string array); what the variable's two-character name is; and, for numeric variables, what the variable's value is.

For string variables, the actual characters in the string are stored elsewhere. The January ***Open-Apple*** (page 4) has picture of this. The variable descriptor for a string, however, does tell Applesoft the string's length and where in memory it is stored.

Because X$ is the first simple variable we define in our program, its descriptor is at the very beginning of the variable tables. A zero-page location known as VARTAB ($69-$70) always points to the beginning of the variable tables, consequently the descriptor for X$ is easy to find. The first of the input-anything routines relied on this.

After calling on NXTCHAR to collect the string and put it in the keyboard buffer, the routine puts the string's length and the address of the buffer into the variable descriptor for X$, as described earlier. When control returns to the Applesoft program, the A$=MID$(X$,1) command is executed. This tells Applesoft to make A$ equal to X$ from its first character onward. The command makes Applesoft dig X$ out of the buffer and move it, as A$, up to the normal string storage location. A$=X$+"" works just as well; A$=X$ does not, however. Pretty tricky, isn't it?

The code this technique needs at 768 looks like this:

```
0300: A2 00     LDX #$00        initialize string length to zero
0302: 20 75 FD  JSR NXTCHAR     call Monitor's INPUT subroutine

0305: A0 02     LDY #$02
0307: 8A        TXA             move string length to A
0308: 91 69     STA (VARTAB),Y  store it in byte two of descriptor
030A: C8        INY               for X$
030B: A9 00     LDA #$00        put address of keyboard input buffer ($200)
030D: 91 69     STA (VARTAB),Y    in bytes three and four of descriptor
030F: C8        INY
0310: A9 02     LDA #$02        note that this only works if X$ is the
0312: 91 69     STA (VARTAB),Y    first variable in the variable table
0314: 60        RTS
```

This routine was written by John Crossley, who went on to become famous in the Apple world as the person who named all the subroutines inside Applesoft. Apple itself has never released the source code to Applesoft, but in an early issue of *Apple Orchard* Crossley named and documented its major useful subroutines. A copy of Crossley's article, "Applesoft Internal Entry Points," was republished in *Call -A.P.P.L.E. in Depth #1: All About Applesoft*, page 51. Crossley's names are now widely used in the Apple II world. At the time Crossley wrote his article most assemblers allowed a maximum of six characters for labels. That's why Applesoft's routines and locations are known today by such concise names as VARTAB.

Crossley's input-anything routine had a bug in it, however, (its strings wouldn't evaluate correctly with the VAL command) and Apple II hackers were soon improving it.

The June 1980 *Call -A.P.P.L.E.* carried an improved "Input Anything Subroutine" by Eric Goez on page 184; *PEEKing at Call -A.P.P.L.E., 1980* had the "Applesoft Input Nearly Anything Subroutine" by Val Golding on page 153; the March-April 1981 *Call -A.P.P.L.E.* had "Input Almost Anything" on page 54; Peter Meyer wrote "The Ultimate Input-Nearly-Anything Routine," which was published in Call -A.P.P.L.E.'s book *All About Applesoft* (released October 1981), page 94; then came Bob Nacon's "The Penultimate Input-Anything Routine" in the January 1983 *Call -A.P.P.L.E.*, page 43.

All of these routines were improvements of Crossley's original. Goez solved the problem with the VAL function; Golding showed how to tack the assembly language portion onto the end of an Applesoft program rather than using the space at byte 768; Meyer, after paying tribute to Crossley's article on Applesoft entry points, showed how to get rid of X$ and how to use the Ampersand command as well as CALL; Nacon showed how to store new strings in the same place previous strings had been stored if the new ones were the same length or shorter—a trick that caused garbage collection to occur less frequently.

Even Uncle Louie tried to get in on the action. Bert Kersey's June 1982 **DOStalk** in *Softalk*, page 195, included an Applesoft routine that called NXTCHAR and then dug the resulting string out of the keyboard input buffer with PEEKs. Unfortunately, it created as much garbage as a GET-loop, only slower. However, it didn't require an extra assembly language routine, nor was it called "Input" anything. For historical reasons, this routine, "Extra Embraced", appears in the *DOStalk Scrapbook* on pages 20 and 21.

Yet another input-anything routine. As the ***Open-Apple*** subscription list grew over the last few months, it became clear that I would have to replace the GET-loop in my mailing label program with something faster if we were going to finish up before the bars closed. First I tried Meyer's "Ultimate" routine and found that, under ProDOS, it issues a carriage return every time NXTCHAR is called. Remember that the ***Open-Apple*** mailing program reads 30 fields per subscriber; that's a lot of carriage returns. Since my program read the file while the printer was on, I soon had a little model of Victoria Falls splashing continuous paper all over the room.

It turned out this problem is very easy to solve if you know the right person to ask about it (Cecil Fretwell, who will be doing a question and answer column for *Nibble* starting later this year—you read it here first), but by now I was mad.

Another problem with any input method that relies on the Monitor's NXTCHAR routine is that any escapes, control-Hs, control-Us, or control-Xs in the file don't make it into the final string. In fact, a control-X in the file will erase everything back to the last return, just as it does when you type it on the keyboard. In addition, if a string is longer than 255 characters, as is often the case with word processor files, the first 255 characters are deleted. Even a GET-loop can bypass this problem.

This was the genesis of this month's "Input Absolutely Anything" routine. It loves commas, colons, escapes, and anything else you want to throw at it. As written, it uses the Ampersand hook, although you could easily change this to a CALL using the information given in Meyer's article. (Don't try to make a Basic.system added-command out of it, however, as PRINT D$;"INPUT A$" will turn your READ off, as all DOS commands do.) As written, the assembly language portion of the routine is loaded at $300, but you could relocate it anywhere.

This input routine isn't meant to be used for keyboard input, although I recommend you try using it that way for demonstration purposes. Because it captures ALL control codes, you can't do ANY on-screen editing. Pressing the back space key will just put a back space character on your screen (a flashing, inverse, or MouseText "H", depending on what video mode and what model of Apple II you are using).

The following program will give you some idea how to use the routine for keyboard input:

```
1000 & A$
1010 PRINT A$
1020 IF LEN(A$) THEN 1000
```

Watch what happens when you press the forward arrow, which creates a control-U. Line 1000 will put this on the screen as a flashing (or inverse or MouseText) U. When line 1010 prints it, however, it will be sent to the screen as a true control-U. If you have 80-column mode on, control-U always turns it off. This creates some interesting effects. To avoid them, try changing line 1010 to:

```
1010 FOR I=1 TO LEN(A$) : PRINT ASC(MID$(A$,I,1));" "; : NEXT : PRINT
```

This will print your string out as a series of ASCII codes and prove that our new routine loves characters of all types.

To read a file with Input Anything, substitute "& A$" (or B$ or whatever$) for "INPUT A$".

For the Basic-only programmers among us, here's a program that will create a machine code file, called IAA.OBJ, that holds the guts of Input Anything. Type it in, run it once, and throw it away (**enter B where you see b; enter 8 where you see B or 8**):

```
 10 REM *** Create IAA.OBJ ***

 20 PRINT "One moment please..."

100 C$="0300: A9 B4 B5  33 20 E3  DF A2 00  20 0C FD  AB 29 7F" : GOSUB 500
101 C$="030F: C9 0D 0B  4B F0 07  C9 20 b0  03 69 40  AB 9B 20" : GOSUB 500
102 C$="031E: ED FD 6B  2B F0 0D  9D 00 02  E8 E0 FF  D0 DD A9" : GOSUB 500
103 C$="032D: 0D 20 ED  FD B6 FD  BA A0 00  D1 B3 F0  02 b0 1F" : GOSUB 500
104 C$="033C: 91 B3 C8  b1 B3 B5  71 4B C8  b1 B3 B5  72 C5 70" : GOSUB 500
105 C$="034b: 6B 90 0D  C5 6F 90  09 BA A2  00 A0 02  20 E2 E5" : GOSUB 500
106 C$="035A: 60 BA 20  52 E4 A2  00 A0 02  20 E2 E5  A0 00 A5" : GOSUB 500
107 C$="0369: FD 91 B3  CB A5 6F  91 B3 C6  A5 70 91  B3 60"    : GOSUB 500

110 FOR I=768 TO 886 : X=X+PEEK(I) : NEXT
120 IF X=15279 THEN 200
130 PRINT "TRANSCRIPTION ERROR. CHECK C$ STRINGS." : END
200 PRINT CHR$(4);"BSAVE IAA.OBJ,A$300,L$77"
210 PRINT "IAA.OBJ IS OK AND HAS BEEN SAVED."
220 END

500 C$=C$+" N D9C6G" : REM S.H. LAM ROUTINE
505 FOR I=1 TO LEN (C$) : POKE 511+I, ASC(MID$(C$,I,1))+128 : NEXT
510 POKE 72,4 : CALL -144 : RETURN
```

Here's an Applesoft routine that will install your new IAA.OBJ file in memory and link it to the Ampersand hook under either DOS 3.3 or ProDOS. Note that it checks the status of the Ampersand hook and will refuse to continue if another program is already using it. Really good Ampersand routines are able to save the contents of the Ampersand hook and daisy-chain to previously-installed routines if necessary. There was neither space nor time for that this month. Run this to install our new input routine:

```
 10  REM *** INSTALL IAA.OBJ ***
100  REM get contents of & hook
110  AMP=PEEK(1014)+PEEK(1015)*256

100  REM determine whether DOS 3.3 or ProDOS is active
120  IF PEEK (PEEK(977) + PEEK(978) * 256) = 76 THEN 150

110  REM ---DOS 3.3---
120  IF AMP < > 65368 THEN 300 : & hook in use if not $FF58 (RTS)
130  X=191 : REM a $BF to stuff at $301 for prompt
140  GOTO 200

150  REM ---ProDOS---
160  IF AMP < > 48643 THEN 300 : & hook in use if not $BE03 (syntax err)
170  X=132 : REM a $84 to stuff at $301 for prompt

200  POKE 1014,0: POKE 1015,3 : REM connect to & hook
210  PRINT  CHR$ (4);"BLOAD IAA.OBJ,A$300"
215  POKE 769,X: PRINT : REM fix $301 for DOS in use
220  PRINT "INPUT.ANYTHING INSTALLED AND READY.": END

300  PRINT "AMPERSAND VECTOR IN USE."
310  PRINT "INPUT.ANYTHING NOT INSTALLED.": END
```

Peekers, pokers, and probers of DOS 3.3 may be interested to know that all of Bert Kersey's **DOStalk** columns and most of mine have just been released in book form. The book is called *The DOStalk Scrapbook*. If you like DOS 3.3 or enjoy dumb jokes you'll love it.

I recommend it to those of you who continue to appreciate the versatility and ease-of-use of DOS 3.3. *The DOStalk Scrapbook* complements the information presented in the venerable *Beneath Apple DOS* in that it gives *examples of how to use* Worth and Lechner's discoveries.

Newcomers, who may find Apple's DOS 3.3 manuals difficult or impossible to buy nowadays, will find it particularly useful. A program disk is available separately that includes all the important stuff from the DOS 3.3 system master, as well as a few surprises (oh, and all 43 of the book's programs, too).

If you were once a reader of *Softalk*'s **DOStalk** column, some of the material in this book may sound familiar to you. But it's been polished and reorganized so the first few chapters are all easy material suitable for beginners (What is DOS?, In the Beginning was INIT, Text Files, Binary Files), while later chapters have more advanced material (using DOS 3.3 from assembly language, using the File Manager and RWTS from Basic, a Sector Read-Write utility).

I recommend you go to every book and computer store in town and ask for it. If they don't have it in stock, make them order it. If you can't wait, you can order it right now from Tab Books, Dept. OA-95, P.O. Box 40, Blue Ridge Summit, Pa. 17214 (800-233-1128). Tab sells the book for $14.45.

After you have these programs working, test things with the keyboard input program given above. Then try it out for reading disk files.

The advantages this routine offers over Applesoft's standard INPUT command are that it can get ANY character; it creates less garbage by storing shorter strings where longer ones had been; and it automatically divides disk strings longer than 255 characters into shorter segments for Applesoft. It does this without any loss in speed. Its disadvantages are that its supporting machine language routine is rather long and that it doesn't allow ANY editing of keyboard input.

Machine language secrets. The complete source code for Input.Absolutely.Anything appears at the end of this article. I present it as meat for the assembly language tigers among our subscribers.

As I indicated earlier, this routine relies heavily on previous work, most of which appeared in *Call -A.P.P.L.E.*. The most significant contribution, the use of the Applesoft routine PTRGET, first appeared in Meyer's article in *All About Applesoft*. Our routine calls PTRGET (pointer get) at byte $304.

PTRGET tells Applesoft that the next item that will appear in the program being executed will be a variable. In our case, this is whatever comes after the ampersand that calls our routine—the A$ in "& A$". If that next item isn't in fact a variable name, program execution will stop with a SYNTAX ERROR.

If it is a variable, PTRGET finds its descriptor in the variable tables and points to the data area of the descriptor with VARPNT—variable pointer—(not to be confused with VARTAB—variable table—which Crossley's routine used). If the variable has never been used before, PTRGET won't find a descriptor and it will make a new one. This is an extremely powerful call.

The tip I picked up from Cecil Fretwell appears at the very beginning of our routine. Both DOS 3.3 and ProDOS look at the PROMPT character to try to determine what is going on. If DOS 3.3 finds a question mark—$BF—being used as the prompt, it figures Applesoft is executing an INPUT command. ProDOS sets the prompt to $84 when an INPUT is executed. To keep Uncle DOS from getting confused about what's going on we must follow this protocol too.

After setting PROMPT and VARPNT, our next task is read a line of text. Since it is NXTCHAR itself that traps control characters, we can't use it. Instead our routine uses the Monitor's equivalent of the GET command, RDKEY. RDKEY puts a cursor up on the screen and jumps to the address stored in the Apple's "input register" or "input link". This always points to an input routine inside DOS.

DOS looks to see if a disk read is in progress. If so, it immediately removes the cursor from the screen. Have you ever noticed a phantom underline appearing on your screen during disk reads? This is RDKEY's cursor (used by NXTCHAR, INPUT, and GET). During disk reads it is on the screen for such a short time that the Apple's video display circuitry usually catches just one line of it.

If a disk read is NOT in progress, DOS will pass the read call on to the current input device. In any event, when the call returns to our own routine, the character read will be in the A register.

We need to determine if the character is a carriage return, which would indicate the end of the line. Since a return can appear as either $0D or $8D, depending on whether it is in the low-value or high-value ASCII format, we save the returned character temporarily in the Y register, then clear its high bit, then test for a return. We'll need the low-value ASCII representation of this character again later, as that's what Applesoft expects to find stored in strings. High-value ASCII characters are not correctly evaluated by Applesoft's VAL command.

Any time DOS sees a call to RDKEY it expects a call to COUT to follow immediately. COUT is the Monitor routine that displays characters on the screen—RDKEY doesn't do this. To satisfy DOS's expectations (and to have what you type appear on the screen), it's necessary to send what RDKEY gives us to COUT.

However, a problem occurs. Some control characters do strange and wonderful things to the screen when sent to COUT. Control-U (the forward arrow), for example, turns off 80-column mode. Since we don't want this to

happen, our routine traps all control-characters other than return and adds $40 to them. This makes them capital letters instead. Depending on what mode your screen is in, they appear as flashing (40-column mode), inverse (80-column mode without MouseText), or MouseText.

After we've sent the character to COUT we recover the low-value ASCII version and check again to see if it is a return. If not, we store it in the keyboard input buffer at $200; add one to the X register, which we use as a string length counter; and go back to RDKEY for another character.

Right in here we also have to check to see if the string is 255 characters long yet. If it reaches that length before we get a return, we pretend we saw one anyhow. This happens in the $328-$330 area of our program. If you would like your longest strings to be less than 255 characters (so you have a little room to add stuff to the end of them), you can POKE a lower value into byte 809 ($329).

Once we get a return, or all the characters we want to handle at one time, we store the string length for later use. Notice that this length includes the carriage return at the end of the line, but the carriage return won't actually be a part of the string. This length value will be stored in the variable descriptor but the true length of the string is actually one less than what the descriptor indicates. This is what Applesoft expects.

At byte $336 we compare the length of our new string with the length of the string this variable had before. If the new string is shorter, we might want to overwrite the old one. Where we store shorter strings depends on whether the old string was actually stored in the normal string storage area or inside the program itself. Consider this line:

```
1000 A$="A BEAUTIFUL HEADING"
```

When a string is defined inside a program like this, it is not moved to the normal string storage area. In order to save space the variable descriptor points right where the string appears in the program. If we didn't check for this and stored new shorter strings here, we would change the program listing. Not a good idea. For example, say we did & A$ and input "UGLINESS IS". The program line would now be:

```
1000 A$="UGLINESS IS HEADING"
```

If you resaved the program at this point you'd have a nice mess on your hands. In order to avoid this problem we check to see at bytes $343-$351 whether the old string is in the normal string storage area. If not, we'll skip down to byte $35B and move the new version up there. If so, we're ready to call an Applesoft routine known as MOVSTR (move string). MOVSTR expects the length of the string to be in the A register, the string's current location in the X and Y registers, and the string's new location in FRESPC. We managed to fill in FRESPC while we were checking to see if the string was inside the program. We take care of everything else in bytes $352-$359 and, if the new string is to overwrite an old one, we are done.

On the other hand, if the new string is longer, or replaces one defined inside the program, another magical Applesoft routine must be called. Again, Meyer's was the first of the works I've mentioned to use it. It's called GETSPA (get space). Our call to it starts at byte $35B. GETSPA expects the length of the string to be in the A register. It makes room in the string storage area for a new string. If no room is available, it will force garbage collection. It automatically sets up the FRESPC pointer MOVSTR needs and returns with the string length still in A.

In this case, we call MOVSTR at bytes $35F-$365. After the string is moved we must correct the variable descriptor for the string's new length and location ourselves. This occurs at bytes $366-$375. Then we're done. So's this article. Here's the source code:

```
                *--------------------------------------
                *     : INPUT ABSOLUTELY ANYTHING
                *     :
                *     : by Tom Weishaar
                *     :     September 1985
                *     :
                *     : a public domain subroutine
                *--------------------------------------

NOTE: Bees in bytes 301, 317, 33A, 33F, 345; all others insects are eights.

0033:           PROMPT .EQ  $33    current prompt is used as a flag by DOSes
006F:           FRETOP .EQ  $6F    new strings are stored here
0071:           FRESPC .EQ  $71    MOVINS' move-to pointer
0083:           VARPNT .EQ  $83    pointer to current variable descriptor
00FD:           LENINS .EQ  $FD    length of current string

0200:           INBUF  .EQ  $200   keyboard input buffer

DFE3:           PTRGET .EQ  $DFE3  finds selected variable and adjusts VARPNT
E452:           GETSPA .EQ  $E452  makes space for string storage
E5E2:           MOVSTR .EQ  $E5E2  moves string from one place to another
FD0C:           RDKEY  .EQ  $FD0C  Monitor read key routine
FDED:           COUT   .EQ  $FDED  Monitor character out routine

                    .OR $300
                    .TF INPUT.ANYTHING

                INPUT.ANYTHING
0300: A9 BF         LDA #$BF        for ProDOS use $84; for DOS 3.3 use $BF
0302: 85 33         STA PROMPT         DOS uses current prompt as a flag

0304: 20 E3 DF      JSR PTRGET      Dig user's variable out of program,
                                       and aim VARPNT at its descriptor
                                       (if first use, make a descriptor)

0307: A2 00         LDX #0          initialize string length to zero
0309: 20 0C FD  .1  JSR RDKEY       get a character
030C: A8            TAY             save it in Y
030D: 29 7F         AND #$7F           clear high bit
030F: C9 0D         CMP #$0D           check for a carriage return
0311: 08            PHP                save result of check
0312: 48            PHA                save character in A (high bit clear)
0313: F0 07         BEQ .2          branch if it was a carriage return

0315: C9 20         CMP #$20        is it any other control character?
0317: B0 03         BCS .2             branch if not
0319: 69 40         ADC #$40        yes--add $40 to make it flashing, inverse,
031B: A8            TAY                or MouseText and slyly slip it into Y

031C: 98        .2  TYA             recover original character RDKEY returned
031D: 20 ED FD      JSR COUT           and print it
0320: 68        .3  PLA             recover version with high bit clear
0321: 28            PLP             remember if it's a carriage return
0322: F0 0D         BEQ .4             branch if it is, we're done
0324: 9D 00 02      STA INBUF,X     otherwise, put it in the keyboard buffer
0327: E8            INX                and add 1 to the string's length
0328: E0 FF         CPX #255           is the string 255 characters long yet?
032A: D0 DD         BNE .1          no, go get another character
032C: A9 0D         LDA #$0D        yes, slip COUT a carriage return even
032E: 20 ED FD      JSR COUT           though one wasn't really in the file

0331: 86 FD     .4  STX LENINS      end of string; save its length
0333: 8A            TXA             move length to A
0334: A0 00         LDY #$0            compare length to the length currently
0336: D1 83         CMP (VARPNT),Y     in the descriptor for this variable
0338: F0 02         BEQ .5          same length or shorter goes to .5
033A: B0 1F         BCS .6          if new string is longer goto .6

033C: 91 83     .5  STA (VARPNT),Y  save new length in variable's descriptor
033E: C8            INY
033F: B1 83         LDA (VARPNT),Y  get current string location from descriptor
0341: 85 71         STA FRESPC         and store at FRESPC for MOVSTR
0343: 48            PHA                save low byte of location
0344: C8            INY
0345: B1 83         LDA (VARPNT),Y  same for high byte
0347: 85 72         STA FRESPC+1
0349: C5 70         CMP FRETOP+1    is string stored in string area?
034B: 68            PLA                (recover low byte)
034C: 90 0D         BCC .6          branch to .6 if not, i.e., if string
034E: C5 6F         CMP FRETOP         is stored in a program statement,
0350: 90 09         BCC .6             don't overwrite it

0352: 8A            TXA             move string length to A
0353: A2 00         LDX #INBUF      put current address of string in
0355: A0 02         LDY /INBUF         X and Y for MOVSTR
0357: 20 E2 E5      JSR MOVSTR      move it to where FRESPC points
035A: 60            RTS             done

035B: 8A        .6  TXA             move string length to A for GETSPA, which
035C: 20 52 E4      JSR GETSPA         makes room in string area, collecting
035F: A2 00         LDX #INBUF         garbage if necessary, and sets up FRESPC
0361: A0 02         LDY /INBUF      put string adr in X and Y for MOVSTR
0363: 20 E2 E5      JSR MOVSTR      move it
0366: A0 00         LDY #$0
0368: A5 FD         LDA LENINS      get new length and store it in variable
036A: 91 83         STA (VARPNT),Y     descriptor
036C: C8            INY
036D: A5 6F         LDA FRETOP      get new address and store it in variable
036F: 91 83         STA (VARPNT),Y     descriptor
0371: C8            INY
0372: A5 70         LDA FRETOP+1
0374: 91 83         STA (VARPNT),Y
0376: 60            RTS             done
```

Ask (or tell) Uncle DOS

Quick, more pie

Here's some footnotes to *AppleWorks Pie* (August, page 57). You can effectively empty the clipboard by copying an empty space to it. Even if you've already defined a custom printer, you can still print formatted text to disk by "adding" one of the listed printers and then choosing "print onto disk" when you're asked for a slot number. The Silentype is a good choice for this.

Debra Hara, whose address you gave on page 61 as the source of free documentation on the internal structure of AppleWorks files, has moved to Mail Stop 3P at Apple. My own *Notes for AppleWorks* will be replaced by a much-enlarged SYBEX book to be released in October. Applied Engineering and Checkmate Technology have expanded the data-base-records limit to 5,114 and 5,100 respectively.

Robert Ericson
Rumford, RI

Competition is a wonderful thing; particularly the price and features competition we've seen in 1985 among the makers of big RAM cards for the IIe. Prices have fallen almost as quickly as the AppleWorks desktop and data base have grown.

AppleWorks users will be interested to know that competition now appears to be heating up among the manufacturers of cards that add a high-speed version of the 65C02 microprocessor to the Apple. Obviously, it takes longer to sort one of these whopping 5,000-record AppleWorks data bases than one of the smaller 500-record files that barely fits on the standard 55K desktop. Card manufacturers think many people who have gotten addicted to AppleWorks' fast response on small files are willing to pay for fast response on large files as well.

The **SpeedDemon** *is such a card; Micro Computer Technologies sells it for $249 (1745 21st St, Santa Moncia, Calif. 90404 213-829-3641). I've had one on loan from M-C-T for about a month. Here's the results of some tests I did with the AppleWorks data base:*

Comparison timings in seconds

AppleWorks data base file with 2,350 records.
(Apple IIe, Sider hard drive, AE RAMworks card)

function	SpeedDemon	standard	faster-factor
load file	28	50	1.8
DA-Find	9	26	2.9
alpha sort	22	54	2.4
numeric sort	25	69	2.8

(Note: AppleWorks sort times depend on the degree to which a file is unsorted. Sorting a file that is already in nearly the specified order is much quicker than the same sort on a file arranged randomly. For these tests, files were shuffled with identical techniques before the timing tests. Also note that the file-loading tests used a hard drive; floppies are much slower on files this large.)

The **SpeedDemon** *fits in any standard slot of an Apple II, II-Plus, or IIe (even in slot 3 on the IIe). When you turn a* **SpeedDemon***ed Apple on, the card takes control and turns the computer's built-in 6502 off.*

The **SpeedDemon** *is then off and running. Unfortunately, however, it can't run at full speed all the time. Lots of routines inside the Apple were specifically designed to execute at the speed of the standard 6502. For example, DOS 3.3 and ProDOS read from and write to floppy disks using routines that contain critical timing loops. If the microprocessor isn't pulsing to the beat of its standard, slower clock, these routines simply don't work.*

To compensate for this, the **SpeedDemon** *carefully watches the softswitches for slot 6. Whenever the address of one of these disk-drive controlling locations appears on the address bus, the* **SpeedDemon** *slows down to normal speed for 50 microseconds. Usually, another reference to slot 6 will appear within that length of time and the timer will be restarted. Consequently, the* **SpeedDemon** *slows down as long as is necessary to read from or write to the disk.*

Optionally, you can also make the card slow down after references to the softswitches for slots 4 and 5. You would do this if you had a second disk drive controller in slot 5. I had to move my ancient pulse-dial Hayes Micromodem to slot 4 to slow down the speed at which it dials the phone. (Even then I had to put an asterisk between all the digits of the phone numbers to keep my modem software from auto-dialing faster than Ma Bell could keep up.)

The **SpeedDemon** *works fine with the Sider hard drive and with revision D and higher of Applied Engineering's RAMworks card (some earlier RAMworks are also supported, ask Applied Engineering for more details). The* **SpeedDemon** *doesn't work with SCRG's* **quikLoader** *because both cards try to take control when the computer is turned on.*

The people at M-C-T tell me they are about to release a new version of the **SpeedDemon**. *With this card you will be able to control speed-up for each slot individually. The newer version will also work with Saturn, Legend, and Prometheus memory cards, which the current version can't do, and will be able to read joysticks and game paddles accurately. Unless you are really good, most games play better with the* **SpeedDemon** *turned off.*

You can disengage the **SpeedDemon** *by turning the computer off, back on, pressing escape within two seconds, then pressing control-reset. Or POKE 49243,0 followed by pressing control-reset turns the* **SpeedDemon** *off from inside a program.*

You can tell when the **SpeedDemon** *is on because the standard Apple beep becomes a chirp You'll also notice that cursors blink and move faster, that screens appear on the screen with more snap, and that microprocessor-intensive tasks take only about half as long or less.*

My biggest reservation about the card is its price. You have to spend lots of seconds sitting in front of a "working" Apple to make $245 seem inexpensive. Let's hope the kind of competition we see in the RAM card arena spreads to manufacturers of speed-up cards. That will bring out more new features and lower prices.

And consider this. Once your computer is operated by a microprocessor on a card, why does it have to be a 65C02? Why not a hi-speed 65802? Does the Apple II have any limits at all?

Enlarging the IIc

I am interested in finding out more about the 256K modification to expand *Apple Works* on the IIc. I would like to know where it goes, if it can be self-installed, if you've tried it, and if you think it's worth it. This is certainly a superior option to me, compared to getting a hard disk drive, since *Apple Works* is the main entity I would need the expanded memory for.

Tom Hays
Wichita, Kans.

Applied Engineering's Z-RAM card for the IIc comes with complete installation instructions. As I understand it—I don't have one (yet)—you must open up the IIc, remove the microprocessor, plug the Z-RAM card into that socket, plug the microprocessor into a socket on the card, and attach one wire for the card to another IIc pin with a clip of some kind.

In addition to your choice of an additional 256K or 512K RAM, you would then have a CP/M computer as well. Z-RAM comes with RAM disk software for ProDOS, DOS 3.3, Pascal, and CP/M; a CP/M-like operating system and CP/M manual; and Applied Engineering's Apple Works Expand *software.*

Among other things, this software allows almost all of AppleWorks to load into memory, which greatly reduces response time—more than a hard disk would. The only problem I see with Applied Engineering's card is that rumors abound that Apple is coming out with its own system for adding RAM to IIes and IIcs and Apple's system probably won't be compatible. (However, it wouldn't surprise me if Apple's scheme was ignored by buyers and software developers in favor of the scheme used by Applied Engineering and other third-party vendors. Apple's scheme will probably prevail, as usual, but it's not a sure thing in this case.)

There's also a rumor that Applied Engineering has a IIc RAM card in the works that will have a clock on it instead of CP/M hardware. Either way, however, if you're happy with the Z-RAM/AppleWorks combination, what might follow tomorrow shouldn't be a major consideration.

II-Plus and extra RAM

How can the II-Plus be expanded to 128K or greater by bank-switching, like the IIe or IIc?

James Landmark
Omaha, Neb.

Memory expansion cards for the II-Plus are available from several different companies. The major problem with these cards is that they all use different bank-switching schemes, none of which is the same as the scheme used on the IIe and IIc. Thus, very little software is available that is capable of using the extra memory, even if you have it.

Should have been selected

We use (and are a dealer for) the PRINT-IT! card manufactured by Texprint that you mentioned in the July issue (page 53-54). It works like the other printer interface card you talked about, but also solves your major reservation—PRINT-IT does beep and display "NOT SELECTED" on your monitor if your printer isn't turned on. Just turn the printer on and the PRINT-IT! card continues immediately.

Texprint now also has *Save-It.* This software, in conjunction with the PRINT-IT! card, will allow you to save any screen into a disk file.

George A Calder
Livonia, Mich.

The control-P conspiracy

I am intrigued by the following behavior of my Apple IIe. When I turn the power on and press control-reset *before* DOS is loaded, I can enter the Monitor, type "1 control-P", and get the contents of any range of memory sent to my printer by typing address1.address2.

But if I enter the Monitor *after* DOS is loaded, this technique doesn't work.

The *DOS Programmer's Manual* lists the values that are supposed to appear in the "Output Register" at $36-$37 when "1 control-P" is entered (page 125). But if DOS is loaded the values at $36-$37 are not what the manual says. Will you kindly enlighten me on this?

Vedula N. Murty
Middletown, Penn.

When DOS is active, the Output Register and its companion Input Register ($38-$39) always point to routines within DOS itself. This is so that every time you type something on the keyboard or print it to the screen DOS will get to see it first. DOS scans all input and output looking for DOS commands.

If you want to route output to your printer when DOS is active, use the PR#1 command, even from inside the Monitor. DOS will see this message as it passes by and execute it. However, even then the actual contents of the Output Register won't change; they'll still point at DOS. DOS 3.3 keeps the true Output/Input Registers at $AA53-$AA56. ProDOS keeps them at $BE30-BE33.

If you use the "1 control-P" command instead of PR#1, DOS will not recognize it as a command. The Output Register will be momentarily changed as described in the manual, but you won't see it because DOS will erase the new value as it hooks itself back up to the Output Register when the next character (the return at the end of the command) passes by.

This process is very complex. For a complete description, see ***The DOStalk Scrapbook****, Chapter 12, "How the System Operates," pages 81-91. If you have a stack of old Softalks, this chapter originally appeared in May 1983, pages 205-210.*

ProDOS and compliers

I have written a rather large programme, running under DOS 3.3, to keep records and a mailing list of several hundred club members. I've done this by reading each member's details into a two-dimensional array, which resides continuously in memory. Although this method uses a lot of space, retrieving details from the array is much quicker than accessing the disk each time a record is required.

The programme works fine. To speed up the various searching and sorting routines it contains, I complied the entire programme using the TASC complier. The compiled programme also works well and of course much, much quicker.

However, both the Basic and complied versions still suffer in speed because of the dreaded Applesoft garbage collector. Thus ProDOS, with its much improved garbage collection, appeared to be the answer.

The uncompiled Basic programme was successfully transfered from DOS 3.3. to ProDOS using CONVERT and garbage collection no longer appears to be a problem. The crunch is, though, that the compiled version, although transferable to ProDOS, does NOT work. It appears that the non-disk routines in the programme all still work, e.g. screen layouts, searching, sorting, etc. However, as soon as disk access is required, the programme stops with text like the following on the screen:

```
OPEN TEST,L170
READ TEST,R1
?
```

This is the type of message one would get if PRINT CHR$(4) had been omitted. I have tested this observation with other shorter programmes that I have written under DOS 3.3, compiled, then transferred to ProDOS. All appear to suffer the same problem—Basic programs transfer ok, TASC-compiled programmes that have no disk reading/writing routines are also ok, but TASC-compiled programmes containing disk routines seem to ignore the PRINT CHR$(4) and come to a grinding halt.

Help (if there is any). If the problem can't be overcome, ProDOS goes to the back of my disk drawer.

W.F. Carey
Athelstone, South Australia

To my knowledge, no Applesoft compilers have been written that work with ProDOS. Under DOS 3.3, assembly language programs could "print" DOS commands just like Applesoft did and DOS would recognize them. Basic.system doesn't support this. This makes a ProDOS-based Applesoft compiler more difficult to write.

Cecil Fretwell has an article in the August ***Call · A.P.P.L.E.*** *(page 19-25) that discusses this problem and offers some possible solutions. He has told me, however, that there is another significant problem with getting a compiler to work with Basic.system. This one has to do with the way Basic.system moves things around in memory when a file is opened and with keeping the ProDOS garbage collector from messing with non-string compiled variables.*

I'm sure someone will solve these problems someday, but it doesn't appear to have happened yet.

The Sider and software

I have seen several references to the Sider hard drive in *Open-Apple*, but not much about software that will work with it. I have a Sider and a II-Plus and have never felt a need to "step-up" to ProDOS. I decided to order a Sider after using IBM-XTs at work and getting spoiled by the speed and ease of operation of a hard disk. My unit arrived 12 days after my telephone order, far ahead of the 30 days I was told to expect. The packing, as you have mentioned, is formidable.

The documentation, alas, is less than the packing. One particular problem, clarified by the friendly people at their 800 number, is the matter of compatibility with other peripherals. The documentation casually mentions that removing all non-essential boards during setup may be necessary. DO IT. My II-Plus and the Sider would NOT work at all during format if I had more than a 16K card in slot 0, floppy card in slot 6, and the Sider card in slot 7.

After formatting (65 DOS 3.3 volumes), I found that my two printer cards (connected to dot matrix and daisy wheel printers) would not work in their usual slots 1 and 2. Something to do with bus timing, or the way I was holding my mouth that day, according to the 800 advisor. After moving my printer cards to slots 4 and 5, I was back in operation.

The next step—what software to use? I use my system primarily for writing so a word processor was my first order of business. I had used the DOS 3.3 version of Apple Writer for more than two years without problems. Now, however, I ran smack into the wall of copy protection.

I finally found a word processor that will do most of what I want—totally within the bounds of the Sider environment. This is the *S-C Word Processor* from S-C Software (P.O. Box 28033, Dallas, TX 75228). The cost is $50 and includes WELL commented assembly source code. This word processor is unprotected, transfers readily to the Sider, uses standard DOS 3.3 commands (including volume number to load and save within the Sider) and has all the usual word processing features.

I have patched my version to cause the QUIT command to return to the Sider menu instead of Applesoft, and to list only text files when the file search option is invoked instead of including binary files such as those created with the old Apple Writer 1.0. I'll pass these patches on to anyone who sends me a self-addressed stamped envelope. Now the only feature I need to print my first novel (when I finish it) is a document-chaining facility to print segments as a seamless whole. Some day, about a year and nine days from now at my present rate, I will figure out the source code well enough to add that feature.

The next program I needed was a spreadsheet to figure how much money I have lost this year by investing in equipment to become a rich and famous author. Once again, copy protection meant that the program I had used for years, VisiCalc, would not work.

The Spreadsheet from A.P.P.L.E. CO-OP, 290 SW 43rd St, Renton WA 98055 fills the bill. The cost is $65 (no source code) and the program uses DOS slot, drive, and volume parameters for loading and saving. You can transfer it to the Sider using the FID-like utility that comes with the drive. It will load my old VisiCalc files and even has features my old VisiCalc lacks. My tax calculations are back in operation.

As well as writing and figuring where all my money has gone, I have done some data base work for a writer's group. I have been using *DB Master* #3 but, you guessed it, copy protection struck again. Fortunately, Stoneware has come up with a version that will work with the Sider. The mail order houses should have this one by now for about $200.

I have *DB Master*'s Sider version 1.6 (they have already promised an update) and have not found it to be really ready for full use. The major drawback is that the program will not actually transfer to the Sider and boot—you must put the program disk in your trusty old floppy drive. It does work, however. It has all the features of regular *DB Master*, has only one tiny bug that I can find (in the file restructure utility), and is almost worth the price. At least my data files are totally on the Sider, eliminating disk swapping for my large files. I would recommend waiting for the next version, however, in the hope that it will run without using a floppy.

There you have it. Three programs that work under DOS 3.3 on a II-Plus with a Sider hard drive. Two of the three are even priced right. Now if I can just find a decent checkbook program.

Tom Smith
1416 NE 98th Ave
Fort Vancouver, WA 98664

I've never used either the **S-C Word Processor** *or* **DB Master***, but I can highly agree with your recommendation of* **The Spreadsheet***. It recognizes and uses the extra memory in most RAM cards, works with many 80-column cards, and is easy for former*

VisiCalc *users to get used to. The same program is also sold under the name* **IACcalc** *by the International Apple Core and many Apple user groups.*

I trust that anyone who has figured out how to add a chain routine to the **S-C Word Processor** *will contact you.*

Time is Money *(Turning Point Software, 11A Main Street, Watertown, MA 02172, 617-923-4441) won't work with a hard drive, saves your data in absolutely unaccessible files, and costs too much. Nonetheless, its speed, flexibility, and capacity more than make up for those things. It's the checkbook/accounting program I use around here and it never ceases to impress me.*

IBM compatible Apple

Lots of people told me it couldn't be done, but I'm running MS-DOS programs on my Apple II with a Rana 8086/2 and storing MS-DOS files on one of the ProDOS partitions of a Sider hard drive. This makes for a fascinating and flexible system. It runs *Lotus 1-2-3* and *Rbase*, as well as a considerable amount of other MS-DOS software (but not *Flight Simulator* or *Sidekick*).

I've found that *X-Basic*, written for the ITT personal computer, works fine with the Rana 8086/2 if programs turn the cursor on with a LOCATE 1,1,1 command. I am interested in corresponding with other Rana 8086/2 users.

Al Grimes
6613 McLean Ct
McLean, Va. 22101

Apple Writer and Applesoft

While browsing through my back issues of ***Open-Apple***, I came upon your recommendation, in the May issue (page 36), to use a word processor to edit Basic programs. I had intended to write you earlier to tell you that I had always thought I was the one who invented that idea; but never mind—I'm grateful to you for spreading the word.

Despite the advantages you mentioned, I had used that technique very little until recently. I have a II-Plus and Videx Videoterm 80-column board, so I needed to boot the Videx pre-boot disk as well as *Apple Writer*. Then, after writing my Basic code, I had to re-boot DOS and EXEC my text file. After checking out the program, I had to reboot the pre-boot and *Apple Writer* to make changes. This was quite a nuisance.

To the rescue came a program with the felicitous name of **OpenAppleWriter**. It allows you to switch back and forth instantly between *Apple Writer* and Applesoft, using only three keystrokes. **OpenApple-Writer** is a product of CondiCom, 436 Berry Dr, Naperville, IL 60540. Their version for DOS 3.3 costs $39, their ProDOS version costs $29, both are $49.

There are other ways in which I use *Apple Writer* in conjunction with Applesoft and DOS. One is to examine and repair sequential text files that were created by a Basic program and that have been messed up. For this purpose, random access files can be converted to sequential files by a Basic program such as the one described by Val Golding in *Call-A.P.P.L.E. in Depth #3: All About DOS*, page 178. To enable the program to continue after a bad record is encountered, I added an ONERR GOTO routine. If the file is really messed up, the program will crash just as you described in January (page 2). Next time I use this program I plan to add your CALL -3288 trick.

Another thing I do with *Apple Writer* (in 80-column mode) is to lay out titles, column headings, and row headings for reports that are to be created with Basic programs. It's easier to visualize how the report will look and to get the spacing and indentation right the first time. I have my Applesoft program READ the heading file as an array of strings and PRINT each string on the appropriate line followed by the computed numbers. To preserve leading blanks and embedded commas and colons, I start each line in the *Apple Writer* file with a quotation mark.

I've also reversed the procedure and used Applesoft to create files to be printed by *Apple Writer*. Once, when I had nothing better to do, I sketched my club's logo on the hi-res screen, then wrote a Basic program to convert hi-res memory to an array of bytes that could be understood by the graphics mode of my printer. With a bit of fiddling, I was able to store this array as a text file that can be loaded into *Apple Writer* and printed as part of other documents, thus adorning club correspondence.

I'm sure you will be pleased to know that I file my copies of ***Open-Apple*** in a three-ring binder of unknown provenance that is imprinted **IBM Data Processing**. Whatever its original contents may have been, its present contents are surely of much greater value.

Paul Nix
Summit, N.J.

Thanks for your tips. According to other information I've seen about **OpenAppleWriter**, *it also does away with the Videx pre-boot disk and allows* **Apple Writer** *to work with hard disk systems under DOS 3.3.*

Another efficient way to edit Applesoft (or assembly language) programs with **Apple Writer** *is to use two computers. This gives you all the benefits of an $8,000 multi-tasking machine, but you have two 80 x 24 windows for your work instead of just one. I boot up* **Apple Writer** *on the IIe in front of me and GPLE or the S-C Assembler on the II-Plus to the right of me. I link the two computers together with an amazing networking device called the floppy disk.*

The program by Val Golding that you mention crunches all the empty space out of a random-access file; the resulting file is not random-access. You can get true random files to load into **Apple Writer** *by filling all the records with blanks before you start saving real data. That way your file will still be random-access but* **Apple Writer** *won't know it and will load it normally.*

Two-column format

Is there a way to achieve a two-column newsletter-type format using AppleWorks and an Imagewriter?

Steve Greve
Augusta, Maine

In **Notes for AppleWorks** *(see August issue, page 61) Robert Ericson suggests the following. Write and edit your text using margin settings that give you the correct width for one column. Clear all tabs, then reset a single tab at the right margin. Use the tab and return keys to put a return at the end of each line. Use open-apple-Z to see what you are doing. Print the resulting document to an ASCII file on disk.*

Make a new, one-category, data base file from scratch with this ASCII file. Create a table-style report and accept the defaults offered. Print this report to a DIF file.

Make a new spreadsheet file from scratch using this DIF file. Widen the spreadsheet's first two columns and use the spreadsheet's copy function to move text from column to column as needed. The spreadsheet becomes a layout surface for your whole newsletter.

I haven't tried this trick, but Ina Levinson wrote it up for the Houston user group newsletter and swears by it. I suspect that codes for bold, underlining, and alternate character widths will disappear, however, which could cause problems.

A way that avoids loss of bold and underlining is to print one column and then roll the paper back manually to print the second column. Start this technique by making your top and bottom margins zero and your page length the actual length you want for your columns. Try the Imagewriter's Proportional-1 type and full JUstification. Write and edit your letter with margin settings that are correct for the first or left-hand column of your newsletter (try an LM of .5, RM of 4.2).

When you are finished editing, use open-apple-K to find page breaks. Change the margin settings for the second page so they are appropriate for the second or right-hand column (try an LM of 4.2, RM of .5) and put an open-apple-O "PH" (Pause Here) code between the new margin settings and the copy. For the third page copy the settings for the first column and add a pause; for the fourth, copy the settings for the second column; and so on.

Print the document. When the printer pauses the first time, manually roll the paper back to the top of the first column. When it pauses the second time, manually roll the paper to the beginning of the second page. It's hard to get the columns exactly lined-up with this technique, but all word processor features are supported and the results are surprisingly good. Sometimes the columns won't come out exactly even because AppleWorks won't put a single line from a multiline paragraph on a page by itself.

A third technique is to simply print the whole thing in a long single column and use scissors and tape/glue/wax to put things exactly where you want them. That's how ***Open-Apple*** *and most other typeset materials are put together and is probably, in the end, the easiest way to get professional-looking results.*

40-column IIe interrupts

We have a system that we have used for some time that uses interrupts. When we first tried to operate this system on an enhanced IIe, however, it wouldn't work. It turns out that on the enhanced IIe, interrupts are routed through Apple's slot-3 80-column firmware. Our problem was that our system runs in 40 columns and we don't normally add an 80-column card. If no 80-column card is found, the enhanced IIe leaves the slot-3 ROM space reading the empty slot and interrupts fail to work.

We solved the problem by pointing the slot-3 ROM space at the 80-column firmware with a poke to 49162 ($C00A).

Larry Swift
Clearwater, Fla.

Correspondence quality phones

I understand you made me world famous recently by putting my phone number in a paragraph about our user group's use of GBBS II and the Sider for a bulletin board system (July, page 53). You neglected to notice our newsletter's OBVIOUS several mentions of the numbers for BOTH our bulletin board (24 hr modem 300/1200, 602-264-3800; name ADAM'S APPLE) and our infoline (24 hr voice 602-277-8511). I don't mind the calls but we have lots of members in ADAM II who can give good advice.

I use a IIe with a Dumpling 64 printer card and an Okidata u92. I use AppleWorks as a data base but find the AppleWorks word processor limited compared to *Apple Writer*. Among other things, it will not stay in correspondence quality. It changes everytime you hit return. Is there a fix for this problem? AppleWorks works beautifully with the Checkmate MultiRam 756K card. They just released the 5,100 record version, WOW!!...it will sort that many records in about a minute and a half.

Jerry Cline, President
Arizona Desert Apple Menagerie II, Inc.
Phoenix, Az.

The AppleWorks word processor automatically turns off bold and underlining at the end of every paragraph and superscripts and subscripts at the end of every line. Thus if "correspondence quality" and "bold" are the same thing, correspondence quality will get turned off everytime a return appears in your file, just as you have experienced.

What you have to do is dive down through AppleWork's printer definition menus until you get to the place where AppleWorks asks for the BOLD END code for your custom printer. Change this code to nothing. After that correspondence quality will stay on.

You may still need a way to turn correspondence quality off, however. To do this, find a character "pitch" or width between 4-per-inch and 24-per-inch that your printer doesn't use. Hardly any printers support that many. You can define unneeded ones to turn special features of your printer on and off.

For example, give AppleWorks the code for turning bold off as the 4-characters-per-inch code. When you want to turn correspondence quality off, turn on 4-characters-per-inch, print a return, then switch back to whatever character size you really want to use. At that point in your document AppleWorks will send the bold-off code and a return to your printer. A slight problem with this trick is that character pitch can only be changed at the beginning of a paragraph, not within one.

The only custom-printer definable codes that can appear within a paragraph are bold, underlining, superscript, and subscript. You can, of course, trick AppleWorks by giving these the codes for something else, but use care or AppleWorks will trick you right back.

IIc serial port noise

We have been reading serial data through a Super Serial Card on a IIe and II-Plus, but when we use the same program on a IIc through serial port 2 we only see garbage. It acts like the parity or data format is incorrect, but using all of the configurations described in the IIc literature the port won't configure to read the data correctly. We took the IIc back to the store where we purchased it. We were told that the configuration had to be set using the IIc utility disk and would only work in ProDOS. We are in the process of rewriting our programs in ProDOS, but it hardly seems reasonable that the ports can't be configured under DOS 3.3.

Jeff Kurtz
Conyngham, Pa.

The information your dealer gave you is wrong. You can reconfigure either of the IIc's ports from inside a program under either DOS 3.3 or ProDOS. To reconfigure port 2, first issue a PRINT D$;"IN#2". You can then PRINT any of several port configuration commands. All of these begin with a control-A and end with a non-control letter. Some have numbers between the control-A and the letter.

A complete list of these commands is in chapter 8 of the Apple IIc reference manual. Many of them—including baud rate, data format, and parity—are the same as the commands for the Super Serial Card, so if you don't have a IIc reference manual use your Super Serial Card manual. (The IIc doesn't support all of the Super Serial Card commands, however.)

When you turn an Apple IIc on, port 2 is automatically configured like this: 300 baud, 8 data bits, 1 stop bit, no parity. If you change this configuration with either the IIc utility disk or [illegible], the new configuration will last until you turn the computer off. Pressing reset or rebooting—even booting a DOS 3.3 disk after configuring with the ProDOS IIc utility disk—won't change the configuration (unless you reboot by turning the computer off and back on).

As mentioned by one of our readers here in June (page 47), early Apple IIcs (serial numbers below D51000) have a hardware bug that can cause the kind of problem you are experiencing. I suspect this is at the root of your trouble. Your dealer is supposed to replace the motherboard in your IIc without charge if you experience this problem—although it sounds like your salesman may be unaware of this. Insist on talking to a technician.

FP.SYSTEM

How do you get rid of Basic.system and still have your ROM Applesoft at hand?

Lambert VanBeers
Tokyo, Japan

I assume you want to get rid of Basic.system but keep the ProDOS kernel. If you don't want the ProDOS kernel either, just turn your computer on without a disk in the disk drive, then press control-reset.

To get rid of Basic.system but keep the ProDOS kernel, use CREATE FP.SYSTEM,TSYS to create a new system file. Enter the Monitor and type 2000:4C 00 E0, then BSAVE FP.SYSTEM,TSYS,A$2000,L3. Rename any other files on this disk that end with ".SYSTEM"; FP.SYSTEM must be the first file in the directory with this suffix.

Now boot this disk and the ProDOS kernel will load and run FP.SYSTEM. FP.SYSTEM simply coldstarts Applesoft, nothing more. Basic.system (and D$ and CATALOG and fast garbarge collection and the page-3 vectors and lots of other stuff) is gone. It's just you, ROM Applesoft, and the ProDOS Machine Language Interface. The first thing you should do when you see the Applesoft prompt is enter HIMEM:48896 to protect the ProDOS global page.

FP.SYSTEM does what you've asked for, but don't take it very seriously. It doesn't adhere to the conventions of a well-written SYSTEM program. The best description of these I have seen is in Gary Little's **Apple ProDOS: Advanced Features for Programmers**, *published by Brady Communications, pages 139-144. I just got this book and haven't gone through it thoroughly, but this section does a terrific job of bringing together in one place snippets of information that Apple scattered through several manuals and tech notes. There may be something missing from the book that I haven't noticed yet, but so far it looks to me like an improved (and cheaper) version of the ProDOS Technical Reference Manual.*

Since you've decided to live without Basic.system, a guide to the Machine Langauge Interface is something you'll require anyhow, so take a look at Little's book.

Single-drive CONVERT

How do you convert DOS 3.3 files to ProDOS files with one drive?

James B Bukowski
St. Paul, Minn.

The IIc system utilities will do it if you have a IIc. On other Apples use the ProDOS CONVERT program. CONVERT, which has probably done more damage to the reputation of ProDOS than anything except the FILER, was designed to work with one drive.

However, an undocumented feature was added to the program to discourage single-drive users. I assume this was because Apple realized DOS 3.3 was the preferable operating system for single-drive systems.

In order to use CONVERT with a single drive, you must first attempt to transfer a file in the opposite direction from the transfer you actually want to do. This initializes some stuff so that transfers in the opposite direction will begin to work. Can you believe it?

Nice letters

I miss the pretty stamps, although I understand why you stopped. I hope this is an indication of a growing subscription list—you deserve success.

T.R. Bainbridge
Kingsport, Tenn.

For such a small publication, ***Open-Apple*** sure packs a *lot* of punch! The two issues I have read so far have had solutions (July's RGB/double-high-res stuff) and hints (August's specify the disk drive as an AppleWorks printer) as well as fixes (August MouseText/CONVERT fix) that will more than pay for the cost of a year's subscription.

Colin Mansfield
Minneapolis, Minn.

We get lots of complimentary cards and letters that we appreciate. Thank you, everybody. I meant to tell you a couple of months ago, but forgot, that this operation started breaking-even in June. ***Open-Apple*** *now has more than 2,000 paid subscribers and is adding a couple hundred or more new subscribers each month. (When the kids refused to lick that many stamps I was forced to start using a postal permit imprint instead.)*

My intention is to continue packing as much useful, relevant, Apple II-related information into 8 pages as possible. I'm grateful that so many of you appreciate ***Open-Apple****'s awesome useful-information-per-ounce-of-paper ratio.*