Open-Apple™

Releasing the power to everyone.

July 1986
Vol. 2, No. 6

ISSN 0885-4017
newstand price: $2.00
photocopy charge per page: $0.15

Shift away from superstition

It's easy to get as superstitious as a baseball player around computers. You may notice, for example, that disks boot better if you scratch your nose with the tip of your elbow before typing PR#6. Some of these superstitions may have an electrical basis, but here's one that doesn't—I read in a user group letter this month that AppleWorks printer setup strings seem to work better if you enter them by holding the shift key down rather than by using CAPS LOCK. Let's stamp this out before it gets serious. On an Apple II there is absolutely no difference between a character entered using the shift key and the same character entered using CAPS LOCK. From inside the machine you can't tell the two apart even if you want to. (It IS important, however, to enter most printer setup strings using capital, rather than lower case, letters.)

ProDOS directory sorters have become popular in the general computer press. *Byte* published one in its June issue (page 117) that contains the same old bug most ProDOS directory sorters have. Don't use it. For more information see the March 1986 ***Open-Apple***, page 2.10. Then tell us why a magazine like *Byte*, which almost never mentions the Apple II to begin with, is writing about ProDOS directory sorters when there are so many more important Apple II technical loose ends to tie up. *Compute!*'s July sorter (page 96) doesn't have the usual bug, but only because it doesn't save the sorted directory back on the disk. It just prints it out to your choice of screen or printer. This and game reviews were *Compute!*'s total contribution to the Apple II literature in July.

Apple's AppleWorks marketers are looking for interesting stories about how real people use AppleWorks. Since ***Open-Apple***'s readers are by and large real, they asked me to ask you to bang out a couple of paragraphs on how you use AppleWorks and send them to Daniel Paul, Mail Stop 3-P, Apple Computer Inc, 20525 Mariani Ave, Cupertino, CA 95014 by July 15.

Uncle DOS recently received a letter headlined TOP SECRET and signed by a character named Sore Throat. The letter divulged that Apple's Software Licensing department has a package available called the Apple II Filecard Tool Kit.

Licensing department documents newly obtained by ***Open-Apple*** describe the package as "four software modules for the 'filecard' (AppleWorks) interface. The 'User Input Routine' object module implements recommended Apple keyboard input for Pascal, Applesoft, and assembly language programs. The 'Console Driver' object module offers fast, efficient 80-column text output and screen control for Pascal, Applesoft, and assembly language programs. A Pascal object 'Console Stuff' unit offers text formatting aids, as well as utilities for overlaying message boxes and Help screens. A Pascal source 'Filecard' unit provides utilities for implementing the filecard interface. Demo source files are also included."

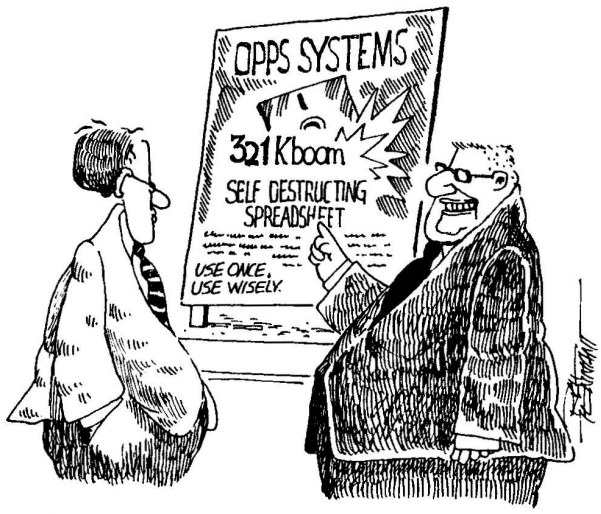
I can find no evidence that Apple has ever mentioned the existence of this tool kit in any of its developer publications. I wish someone would explain to me why Apple seems so dedicated to trashing the filecard interface in favor of the mouse desktop, which is more complicated for both users and programmers. There are more copies of AppleWorks being used today than there are copies of the Macintosh itself. The filecard interface is intuitive, understood by people all over the world (even computer salespeople), and requires only two hands. Nonetheless, during the last two years I have never seen or heard Apple encourage anyone to use it, with one exception. The mouse desktop, on the other hand, is shrilly promoted wherever Apple gathers developers.

The one exception is in *Apple II Human Interface Guidelines*. This manual includes a 14-page chapter on the filecard interface. The more complicated mouse interface requires a 55-page chapter.

Apple's Software Licensing department collects fees from people who use Apple-created software in their own software packages. In exchange for paying the fees you get the right to include Apple's software in an unlimited number of copies of your own package. You also get free updates to the licensed software. With some items you get a copy of the licensed software, technical documentation not available elsewhere, or both.

To get in touch with these people, call 408-973-4667 or write Software Licensing, Mail Stop 28-B, Apple Computer Inc, 10101 N DeAnza Blvd, Cupertino, CA 95014. Here is a complete list of the Apple II products currently available for licensing. Items marked with "*" will be provided ("*R" means provided by request only), items marked "U" are unsupported:

DOS 3.3	$50	*R	U
FID	$50	*R	U
FPBASIC/INTBASIC	$25	*R	U
DOS TOOL KIT	$50		U
ProDOS/Basic.system/			
FILER/CONVERT	$50		
Basic.system source code	$100	*	
ProDOS disk formatter	$50	*	
ProDOS USERS DISK	$50		U
(German)		*	U
(French)		*	U
(Italian)		*	U
ProDOS BACKUP II	$35		U
Filecard Tool Kit	$50	*	
Desktop Tool Kit	$50	*	
Printer Tool Kit	$50	*	
6502 Std Apple Numerics (SANE)	$50	*	



"WE'RE VERY PROUD TO OFFER SOFTWARE INCORPORATING THE LATEST ADVANCE IN COPY-PROTECTION."

SuperPilot	$50		U
SuperPilot Library	$25	*	U
Pascal operating system	$100		
ProDOS access unit	$50	*	
Volume manager unit	$50	*	
Pascal formatter 1.3	$25	*	
Pascal filer 1.3	$50		
Attach tools 1.3	$50	*	
Pascal short graphics unit	$25	*	
Pascal 1.2 48K runtime pkg	$50	*	
Pascal 1.3 64K or 128K runtime	$50	*	
Pascal 1.3 treesearch, idsearch	$50	*	

If you're wondering whether you have to be a "third-party developer" to license these packages, you'll be glad to hear that Apple seems to define "developer" as someone who has licensed something from them. If you'd like a copy of the Filecard Tool Kit, for example, call and ask for copies of Apple's Software License Agreement. Be prepared to provide a business name—these folks know me as Tom Weishaar Productions, for example. They'll send you some long legal forms and "Exhibit B," which is a list of the packages available for licensing. Sign the forms, send them back with a check, and they'll send you disks and documentation.

The most interesting stuff is the three tool kits, the SANE package, and the Pascal collection. I'm not sure what's in the Filecard Tool Kit at the moment; when I ordered it together with the Desktop Tool Kit I got the Printer Tool Kit instead. (A quick phone call and the right package is on the way.)

Meanwhile, I can tell you the Desktop Tool Kit consists of eight disks—four assembly language/Applesoft and four Pascal. Two of each set of four disks includes routines for doing the mouse desktop on the 80-column text screen, the other two have routines for doing the mouse desktop on the double-high resolution graphics screen. In addition to the eight disks, you get photocopied versions of the following manuals: *Apple II Human Interface Guidelines, Developer's Handbook for the Apple II MouseText Tool Kit, Mouse Graphics Tool Kit: External Reference Specification,* and *Apple II Graphics Primitives Handbook.* This is an immense amount of material for $50, especially from the tightwads at Apple.

The Printer Tool Kit consists of two ProDOS disks that are filled with assembly-language color-and-monochrome-graphics printer-drivers for the Apple Scribe, Imagewriter, and Imagewriter II printers. It includes a photocopied manual called the *Developer's Handbook for the Apple II Printer Tool Kit.* The drivers support control of picture size and aspect ratio; RGB 8- and 16-color, NTSC 8- and 16-color, and high-resolution and double-high-resolution monochrome graphics; screen pre-view and cropping; and page positioning and rotation.

The SANE (Standard Apple Numerics Environment) package consists of three disks—one ProDOS and two Pascal—full of assembly language routines and a printed copy of the *Apple Numerics Manual.* SANE is one of the first widely-available products that conforms to a binary floating-point arithmetic standard called IEEE 754. It provides for four types of variables with 7, 15, 18, and 19 significant decimal digits. (Applesoft provides nine significant digits, Apple Pascal six.)

None of the manuals listed here was included in ***Open-Apple's*** March list of technical documentation available from Apple (page 2.11).

More technical support for anyone who wants it (and can afford it) is available from Apple in the form of subscriptions to the Apple II Technical Notes. ***Open-Apple*** published a list of the current tech notes in March (page 2.11) and updated it last month (page 2.34). Apple is now selling subscriptions to the notes to anyone who's interested. A complete set of 1985's notes is $45 (all active notes were revised in 1985, so this is the equivalent of a complete set of back issues); a subscription to this year's notes, which are issued every other month, is $25. Subscriptions are available from the Apple Computer Mailing Facility, 467 Saratoga Ave, Suite 621, San Jose, CA 95129 (408-988-6009). Also available, for a mere $75 each, are *Inside Apple Talk* and *Inside LaserWriter.*

Apple's lawyers were stung by their first big defeat in May, when Australia's High Court ruled that, prior to the amendment of the Australian Copyright Act in 1984, software in ROM was not eligible for copyright under Australian law. The court, which is the highest in Australia, overturned a decision of the full Federal Court, which had upheld Apple's claim of copyright infringement against Computer Edge Pty Ltd, importer of the Taiwanese-made Wombat Apple II-compatible into Australia.

The earlier decision of the full Federal Court, which put Computer Edge out of business, had itself overturned an original Federal Court decision in this long legal battle, which Apple lost in December 1983. That decision led the Australian government to rewrite its Copyright Act in 1984. The new act gives all software, even that written before 1984, full copyright protection.

Thus Apple's loss is largely academic in Australia. It could be more significant in other Commonwealth countries, such as Canada and Hong Kong, however, that have not yet rewritten their copyright laws to specifically include software. Cases such as Australia's can be cited in courts throughout the Commonwealth when the underlying laws are similar.

Meanwhile, in other legal action in May, Apple won a lower court copyright decision against Canadian importers of Apple II-compatibles. In that case the importers conceded that source code can be copyrighted, but argued that ROM was not a suitable material for a literary work. The court decided that the wording of the copyright act was broad enough to include silicon chips. It isn't yet known what effect this case may have on an ongoing update to Canada's copyright law.

Question: why do all these suits focus on whether software in general is copyrightable rather than on whether the Apple II Monitor in particular was placed in the public domain by Steve Wozniak prior to being copyrighted by Apple Inc?

Picking Up Applesoft

Exercises in self-modification

Back in March Uncle DOS got a question about how to capture a user's formula interactively on the keyboard and enter it into a running program (page 2.16). Uncle didn't know how, but referred to four articles in *Call - A.P.P.L.E.* and *inCider* that explained how it could be done.

In April ***Open-Apple*** carried a long article on breaking the 48K memory limit of Applesoft using a RAMdisk (page 2.17-2.21). This article went into great detail on the structure of Applesoft programs as they appear in memory. Among other things, it showed how to split Applesoft programs to avoid the graphic screens and how to overlay program segments.

Then, in June, ***Open-Apple*** built upon the April article by showing how to examine the value of all currently defined variables. The program presented in that article, using April's information on the structure of an Applesoft program in memory, modified itself. It introduced a routine that could identify the memory location of any program line. Once that is accomplished, it is a fairly simple matter to POKE changes into the line.

This month, we'll build upon the June article and answer March's question about getting a user's formula into a program. We'll also show how to hide short assembly language routines inside the body of an Applesoft program.

Self-modifying Applesoft. Let me begin by reminding you that data processing professionals abhor self-modifying code. Don't discuss what you learn here in polite company or during any job interviews.

In last month's article, I showed you a program that could make fixed-length changes to itself. For example, we had the line:

```
37824 : PRINT N$;" = ";XX$
```

Our goal was to change the "XX$" to the actual name of the variable we wanted to print—maybe "OZ%", for example, or "OZ " (with a space after the Z to denote a floating-point variable). This was easy because we always knew ahead of time exactly how many characters were involved. When you start letting people enter command strings on the keyboard, however, you don't know ahead of time how long the string will be.

Somehow, we have to write a program line that can be modified later and that reserves the maximum amount of space we'll ever need. How about this one:

```
370 :REM**************************************************************
```

Last time I counted there were 60 asterisks in that line, which gives us a 60-byte command line to work with. If the user enters a 30-byte command, however, we also need a way to tell Applesoft to ignore the extra asterisks at the end of the line. We can do this by adding a colon and a REM to the end of the user's command. For example, if the user wants to insert the formula FOR

X=0 TO 30 : Y=1000 * 1.08^X, then line 370, after modification, should look something like this:

```
370 FOR X=0 TO 30 : Y=1000 * 1.08^X : REM*****************************
```

Now all we have to do is find the location of line 370 in memory. Here's the routine we used last month to do this. You call this routine after putting the number of the line you want to find in L. It returns the address of that line in ACL. Line 100 was numbered 37700 last month; I had to move it to the beginning of the program to get everything to work:

```
100 DEF FN PK(ADR)=PEEK(ADR)+PEEK(ADR+1)*256

37990 REM * Find line number L                *
37991 REM *   ACL=address of current line     *
37992 REM *   ANL=address of next line        *

37993 REM set ACL to start of program (TXTTAB)
37994 ACL=FN PK2(103) : REM address of current line
37995 ANL=FN PK2(ACL) : REM address of next line

37996 REM find line number L
37997 IF L=FN PK2(ACL+2) THEN RETURN
37998 V=FN PK2(ANL) : IF V>0 THEN ACL=ANL : ANL=V : GOTO 37997
37999 PRINT "There's no line ";L;" in this program." : STOP
```

Unfortunately, simply collecting a command string with an INPUT statement and POKEing it into line 370 doesn't work. The reason it doesn't work is that Applesoft expects command words such as PRINT and IF to appear as tokens, as discussed in April (page 2.18).

When I found this out, I read the four articles Uncle DOS recommended in March to see how those authors had solved the token problem. The most helpful article, as is often the case, was the one by Cornelius Bongers. Bongers identifies the routine inside Applesoft that converts program lines typed on the keyboard into tokens. After reading the four articles, I decided the best way to proceed was to use a smidgen of assembly language.

The Applesoft tokenization routine is at $D56C. Under normal conditions, it converts the ASCII contents of the keyboard input buffer ($200-2FF) into a tokenized program line. The converted line is never longer than the original, so the converted line is stored right over the top of the original in the buffer. The routine expects the address of the keyboard buffer to be stored at byte $B8-$B9 (184-185) and requires a couple of other bytes to be initialized. The following routine, which we'll call PARSE, saves and restores the contents of $B8-$B9, does the other required initialization, and executes $D56C:

```
8E4:  A5 B8     LDA $B8
8E6:  48        PHA         save bee-eight and -nine on stack
8E7:  A5 B9     LDA $B9
8E9:  48        PHA
8EA:  A2 01     LDX #1      avoid having a zero in the program
8EC:  CA        DEX            make a zero
8ED:  86 B8     STX $B8        store at bee-eight
8EF:  CA        DEX            make it $FF for PARSE
8F0:  A0 02     LDY #2
8F2:  84 B9     STY $B9     $B8 now points at keyboard buffer at $0200
8F4:  84 13     STY $13     initialize data flag
8F6:  A0 04     LDY #4      initialize Y
8F8:  20 6C D5  JSR $D56C   execute PARSE
8FB:  68        PLA
8FC:  85 B9     STA $B9     restore adr at bee-eight to previous value
8FE:  68        PLA
8FF:  85 B8     STA $B8
901:  60        RTS
```

Assume we have this routine in memory and have told the variable PARSE its address. We can now have the user enter a desired formula using INPUT and convert it to tokens like this:

```
300 L=370 : GOSUB 37990 : ADR=ACL+4 : REM find line that will be modified
310 INPUT "ENTER APPLESOFT CMD: "; CMD$
320 CALL PARSE
```

ADR points to the colon in line 370. The colon is four bytes beyond the beginning of the program line (two bytes for the next-line pointer, two bytes for the line number), thus the ADR=ACL+4 in line 300. Now let's move the tokenized line from the keyboard input buffer into line 370:

```
325 ROOM=60 : REM this is the number of asterisks in line 370
330 FOR I=0 TO ROOM-1
335 : V=PEEK(512+I) : I1=I
340 : IF V=0 THEN I=ROOM : GOTO 350
345 : POKE ADR+I,V
350 NEXT
```

PARSE will leave a zero at the end of the tokenized line. Line 340 looks for this zero and gets us out of the loop when it finds it. If a user enters a command that takes up more than 60 bytes, we need to print an error message:

```
355 IF V<>0 THEN PRINT "FORMULA TOO LONG" : PRINT : GOTO 310
```

And don't forget that we always need to add ":REM" at the end of the user's command:

```
360 POKE ADR+I1,58 : POKE ADR+I1+1,178 : REM ":REM"
```

A hidden beauty of this trick is that the user can include commas and colons in the formula, even though INPUT won't accept them and will print an EXTRA IGNORED message. The reason for this is that everything the user types goes into the keyboard buffer, whether Applesoft ignores it or not. Notice that we never actually do anything with the CMD$ that we collect with INPUT in line 310. Instead, we let PARSE work directly on the contents of the keyboard buffer.

What if the user types a command line that has a SYNTAX ERROR? Let's trap it and let the user re-enter the line:

```
365 ONERR GOTO 400 : REM syntax error trap
370 :REM*************************************************************
380 POKE 216,0 : PRINT : GOTO 310 : REM clear onerr, restart

400 IF PEEK(222) <> 16 THEN PRINT "ERROR #";PEEK(222);
      " IN LINE ";PEEK(218) + PEEK(219)*256 : END
410 PRINT "THIS CMD HAS A SYNTAX ERROR."
420 CALL -3288 : GOTO 380 : REM clear stack--see Jan 85, page 2
```

Line 380 sends us back to line 310 for another command. Let's use CMD$ after all to give the user the option to quit by entering an empty line. And when the user quits, let's have our little demo program list the formula we've placed in line 370, as well as some other stuff we're going to put in line 230:

```
315 IF LEN(CMD$)=0 THEN LIST 230: LIST 370 : END
```

Embedding machine code in Applesoft REMs. To finish our little program, we have to figure out some way to get our PARSE routine into memory. In the past, ***Open-Apple*** has always tucked such routines into the free memory in page three. Because this area is used for so many different purposes, however, you virtually have to reload code there every time you want to use it. You can't depend on a piece of code still being there from the last time you used it.

This is no good. One way around the problem is to actually store your machine language code in REM statements. If you do this, you definitely don't want Applesoft to attempt to execute the program line, so leave a REM at the beginning to make Applesoft ignore it. Use exactly as many asterisks as there are bytes in your assembly language routine:

```
200 L=230 : GOSUB 37990 : PARSE=ACL+6 : REM find line that will be modified
230 :REM*******************************
```

This time we want line 200 to point at the first asterisk in line 230, rather than at the colon as we did with line 370. Thus the PARSE=ACL+6 in line 200. The colon and REM are already tokenized and thus fill one byte each.

There are a number of techniques for inserting assembly language code into memory from Applesoft, such as READ-DATA loops and so on. Here we'll use my favorite, the Lam technique, which has been discussed in ***Open-Apple*** a number of times (pages 12, 23, 77, 2.16).

One problem, however, is that we don't know ahead of time the exact address where the routine is to be placed. Normally with the Lam routine, you would put an RTS at $300, for example, with the Monitor command string "300:60". Rather than messing around converting Applesoft's decimal number to hexadecimal and inserting it at the beginning of the command string, let's POKE the address in bytes $40 and $41 (64-65), which is where the Monitor stores the "next changeable location." Then, rather than beginning the command string with an address, we simply start with a colon, like this:

```
210 POKE 65,PARSE/256 : POKE 64,PARSE-(PEEK(65)*256)
220 C$=":A5 B8 48  A5 B9 48  A2 01 CA  86 B8 CA  A0 02 B4
        B9 B4 13  A0 04 20  6C D5 68  85 B9 68  85 B8 60" : GOSUB 500
```

The rest of the Lam technique can be found in the listing at the end of this article at lines 500 through 530.

There are two tricks to writing assembly language routines that are to be stored inside Applesoft programs. First, of course, the routine must be relocatable. It can't refer to any locations within itself except with relative

Ask (or tell) Uncle DOS

The system works

I took a copy of your answer to my April letter (page 2.23) to the technician at the local Computerland. The letter was about the trouble I've been having getting disks to boot since buying an Apple color monitor.

When the technician checked my machine, he discovered that you were right; a shield was missing from my disk drive, and he replaced it. All of the programs that would not boot properly before work well now. As an added bonus, the shield also solved some difficulties I have experienced in making copies of some disks.

Thank you for your help. Even though most of the discussion in your pages is way over my head, I will remain a loyal subscriber.

Bernice Eaton
Northridge, Calif.

My subscription is up and six months ago I thought I wouldn't renew. Your paper is for the computer sophisticate. Your paper is too far above the layman. Then little by little I started getting a glimmer of what was going on. I've been going over all the back issues, and I'm UNDERSTANDING them. Gee, what next? I can learn what has heretofore seemed esoteric jargon. Thanks.

Harry Charles
Nederland, Texas

Understanding the Apple II requires neither superior intelligence nor a chip-like personality. ***Open-Apple*** *really is dedicated to "releasing the power to everyone." It does take awhile to absorb enough of the language and of the concepts to get started, but thousands of perfectly normal people have learned enough to make their Apple IIs jump through hoops of fire. Remember—the more you read, the more you understand; the more you understand, the more you understand.*

Roots

What is the nature of the mysterious appendix that appears when you LIST the Integer Basic APPLEVISION program found on old DOS 3.3 system master disks?

Where is the machine code with the dancing man and the little tune?

Len Lipschutz
Jersey City, N.J.

In the Olden Dayes, when I myself was just an Apple II beginner, APPLEVISION was a maddening program. I wanted to write programs like that one, but Apple's manuals had no clues whatsoever about how to proceed.

Now I know enough at least to tell you that you spoiled the fun by answering your own question; the "mysterious appendix" IS the machine code. My understanding is that it is fairly easy in Integer Basic to attach machine code routines to the end of programs, although I still don't know how to do it. Back in the days of cranky cassette tape, attaching the machine code to the program made loading and saving a great deal easier.

If you're really curious, the January 1984 issue of ***Apple User*** *has an article that shows how to capture the hi-res character generator buried in APPLEVISION for your own use.* ***Apple User*** *is published by Database Publications Ltd., Europa House, 68 Chester Road, Hazel Grove, Stockport, SK7 5NY, U.K. (061-480-0171). Annual subscription rates are 15 pounds in the U.K., 23 pounds in Europe, and 38 pounds elsewhere (airmail).*

Double page 2

Oh yes there IS page 2 double-high res! When 80STORE is on, the PAGE2 softswitch flips between main and auxiliary memory, just as you demonstrated last month (page 2.40). However, when 80STORE is off (POKE 49152,0), the PAGE2 switch flips between double-high res pages 1 and 2. I got this from Roland Gustafasson, who's motto is "Don't believe everything you see in the manuals."

David Eisler
Littleton, Colo.

Well, to quote what I said last month, "there is no more obscure arena in the Apple II world than double-resolution." My apologies to the author of Apple's RGB manual, who got it right after all.

MacroWorks' mouse

I've been using *MacroWorks* for several weeks. My IIc mouse works fine when I am in the *MacroWorks* program; however, it does nothing when I am in AppleWorks. The *MacroWorks* manual says nothing

---

branch instructions. Second, things work best if you avoid having any zero bytes embedded within the code. For example, LDA #0 becomes A9 00 in actual machine code. If you embed this in a REM statement and then edit the program, the Applesoft editing genie will see that zero, assume it marks the end of a program line, change some next-line pointers around, and create the conditions for a certain SYNTAX ERROR the next time the program is run.

Notice the trick used in the fifth and sixth lines of our PARSE routine to avoid a zero.

If you can't avoid zero, all is not lost. The limitation is that you must take great care to edit your program only when there are asterisks in the REM statement. Don't attempt to edit or save the program after RUNning it.

If you can avoid zeros, on the other hand, you can actually SAVE the program with the assembly code embedded in the REM statement. Before saving you can delete lines such as 210, 220 and 500 through 530. You can leave them out of your program from that point on. Note that you will will still need a line like 200, however, to figure out where the routine has ended up.

Here's a complete listing of this month's tricks:

```
*--------------------------------
*     : SELF.MODS DEMO
*     :
*     : by Tom Weishaar
*     :     July 1986
*     :
*     : a public domain program
*--------------------------------
```

```
100 DEF FN PK(ADR)=PEEK(ADR)+PEEK(ADR+1)*256

200 L=230 : GOSUB 37990 : PARSE=ACL+6 : REM find line that will be modified
210 POKE 65,PARSE/256 : POKE 64,PARSE-(PEEK(65)*256)
220 C$=":A5 B8 48  A5 B9 48  A2 01 CA  86 B8 CA  A0 02 84
        B9 84 13  A0 04 20  6C D5 68  85 B9 68  85 B8 60" : GOSUB 500
230 :REM*******************************
```

```
300 L=370 : GOSUB 37990 : ADR=ACL+4 : REM find line that will be modified
310 INPUT "ENTER APPLESOFT CMD: "; CMD$
315 IF LEN(CMD$)=0 THEN LIST 230: LIST 370 : END
320 CALL PARSE
325 ROOM=60
330 FOR I=0 TO ROOM-1
335 : V=PEEK(512+I) : I1=I
340 : IF V=0 THEN I=ROOM : GOTO 350
345 : POKE ADR+I,V
350 NEXT
355 IF V< >0 THEN PRINT "FORMULA TOO LONG" : PRINT : GOTO 310
360 POKE ADR+I1,58 : POKE ADR+I1+1,178 : REM ":REM"
365 ONERR GOTO 400 : REM syntax error trap
370 :REM*************************************************************
380 POKE 216,0 : PRINT : GOTO 310 : REM clear onerr, restart

400 IF PEEK(222) < > 16 THEN PRINT "ERROR #";PEEK(222);
      " IN LINE ";PEEK(218) + PEEK(219)*256 : END
410 PRINT "THIS CMD HAS A SYNTAX ERROR."
420 CALL -3288 : GOTO 380 : REM see Jan 85, page 2

500 C$=C$ + " N D9C6G" : REM space required before and after N
510 FOR I=1 TO LEN(C$)
512 : POKE 511+I, ASC(MID$(C$,I,1))+128
514 NEXT
520 POKE 72,4 : CALL -144
530 RETURN

37990 REM * Find line number L                *
37991 REM *   ACL=address of current line      *
37992 REM *   ANL=address of next line         *

37993 REM set ACL to start of program (TXTTAB)
37994 ACL=FN PK2(103) : REM address of current line
37995 ANL=FN PK2(ACL) : REM address of next line

37996 REM find line number L
37997 IF L=FN PK2(ACL+2) THEN RETURN
37998 V=FN PK2(ANL) : IF V>0 THEN ACL=ANL : ANL=V : GOTO 37997
37999 PRINT "There's no line ";L;" in this program." : STOP
```

about the mouse support you described (June issue, page 2.33). How do I get it to work?

Robert C. Moore
Laurel, Md.

Beagle Bros has a free upgrade for you. Send them the proof of purchase tab from the back cover of your manual and they'll send you a new disk and documentation.

*Randy Brandt, **MacroWorks** author, tells me the most creative use of macros he's heard of so far was from a user who wrote macros to change his custom printer settings. These macros allowed him to work around the AppleWorks limit of a single custom printer. The macros take 18 seconds to completely redefine the custom printer.*

SuperCalc 3a defended

I'm curious to know the reasons behind your personal vendetta with SuperCalc 3a. You've panned the program twice now (January 86, page 98; May 1986, page 2.28).

I can't believe we're talking about the same program. SuperCalc 3a uses slash commands familiar to every user of VisiCalc or its clones. It includes all the numeric functions AppleWorks left out, on-line context-sensitive help screens, a superb graphing function, and built-in sideways printing. Plus it's unprotected, automatically recognizes the extra memory on my RamWorks and Apple memory cards, and runs from floppies, UniDisk 3.5, hard disk, or RAMdisk.

I suggest you take another look.

Marc S. Renner
St. Paul, Minn.

I think it's important to realize I'm cynical, grumpy, hot, and tired when I test new products. Consequently, even I don't consider my opinion the last word on a product's quality or importance. I appreciate it when those of you who disagree with my impressions say so.

*My problems with the **SuperCalc** interface center on the unusual use of the escape key when "pointing" to cells while building formulas and the way the arrow and return keys work. Other than having reservations about the interface, however, I agree with you—as I said in May, the program is the most powerful Apple II spreadsheet I've used. I'll take another look at it next time I have some real number crunching to do.*

LDA (83,Y)???

In the SU2.OBJ letter in your May issue (page 2.32), shouldn't the LDA (83,Y) instructions be LDA (83),Y?

Dick Ellicott
Baltimore, MD

Yes. And that's how they appeared in Ruth's original letter, too. We're now investigating how those closing parentheses managed to move themselves to where they didn't belong.

Whence come these pauses?

One of my programs uses three nested FOR-NEXT loops to deal with a sequential text file. The loops cause the disk drive to start and stop several times when transferring data between the disk and memory. In some cases pauses last as long as five seconds.

As I had not seen this happen before, I undertook a scientific investigation to determine the cause. Of course I realized that with my first loop 1 to 12, the second loop 1 to 9, and the third loop 1 to 10 that there were a lot of loose strings running around in the system. Isn't it 12 * 9 * 10 (or something like that)?

Rather than play around with that complicated program, I wrote a simple one to determine what the break point was for the disk drive to run continuously or to play stop and go. I created a file that consisted only of a list of numbers from 1 to 5000. I discovered that when X <= 88 the drive ran only once, but that when X=89 the drive would run, stop, then run again. Beginning at X=156 the drive would stop twice. At X=220 the drive stopped three times. The first incremental stop happened at 88, the second incremental stop 67 numbers later, and the third incremental stop 64 numbers after that.

At this point it was obvious there was no rhyme or reason to the progression of stoppings, so I gave up. Why is my lazy IIe telling the disk drive to take a rest every now and then while it chews on the data? Is my faultless programming at fault?

Barney Woodruff
Camp Springs, MD

When you PRINT a number to a text file, the text file actually gets one character for each digit in the number, plus a carriage return at the end. So, for 1 through 9, you print two characters (a digit and a carriage return) per number. For 10 through 99, you print three characters (two digits and a carriage return).

There were 88 numbers written before the first drive access occurred. That's 1 through 9 (2 characters each), plus 10 through 88 (3 characters each), or a total of 255 characters.

*Then you printed 89 through 155. This is 11*3 (for 89-99) plus 56*4 (for 100-155)—257 characters.*

*Then you printed 156 through 219. This is 64*4 or 256 characters.*

When you print characters to a file, DOS takes the characters and stores them in an area called a DOS buffer, which holds the same number of bytes as one storage unit on the disk (256 for DOS 3.3, 512 for ProDOS). Only when the DOS buffer is full does DOS actually turn on the disk drive and store your numbers on your disk.

When you are reading from the disk, the opposite occurs. DOS gets a bufferfull of digits from the disk, then feeds them one number at a time to your INPUT statements.

The pauses occur because Applesoft can't process the bufferfull of digits as fast as DOS can fill the buffer. So DOS has to stop and wait while Applesoft works. This usually happens only with text files, because binary and Applesoft files are loaded into memory in mass.

You'll find that some Applesoft programs and most assembly language programs can process a buffer full of bytes fast enough to keep the drive from turning off. With Applesoft, this would happen with programs that use longer strings and require fewer PRINT or INPUT statements to process a whole buffer.

The reason the first two numbers you came up with don't match (255 characters versus 257) is that the three characters of 89 (two digits plus the carriage return) were split between the first two sectors written. After your loop has PRINTed the numbers 1 though 88, 255 bytes of the DOS 3.3 buffer have been filled. The 8 digit of 89 fills the 256th and final byte, so Uncle DOS writes the buffer to the disk. Applesoft then starts filling the now-empty buffer with new data, starting with the 9 digit and the carriage return remaining from 89. This takes so long that Uncle DOS gets tired of waiting and turns the drive off.

The bytes in the final DOS buffer don't get saved on your disk until you issue a CLOSE command (under ProDOS, a FLUSH command also does this). This is why it is very important to always CLOSE files.

Because your pauses occurred where they did, incidentally, we can tell you were using DOS 3.3. If you had been using ProDOS to run this test you would have encountered pauses only half as often, because a ProDOS buffer is twice the size of a DOS 3.3 buffer.

*The slow DOS 3.3 garbage collector may also be involved in the length of the pauses you experienced with your original program. For possible fixes, see the **Open-Apple**s for January 1985, pages 4-5; March 1985, pages 17-19; October 1985, page 76; and May 1986, page 2.31-32.*

Basic.system tricks

I knew you were tired of the same old ProDOS boot filename STARTUP, so I wrote this little routine to enable you to change it to any legal ProDOS filename of seven characters or less:

```
10 D$=CHR$(4) : INPUT "NEW BOOT FILENAME: "; F$
15 IF LEN(F$) > 7 THEN PRINT "TOO LONG" : GOTO 10

20 PRINT D$;"BLOAD BASIC.SYSTEM,TSYS,A$2000"
25 V=PEEK(8192)
30 IF V=169 THEN A=8677 : REM version 1.0
35 IF V=76  THEN A=8198 : REM version 1.1
40 IF A=0 THEN PRINT "Version unrecognized" : END

50 POKE A, LEN(F$)
55 FOR X=1 TO LEN(F$)
60 : POKE A+X,ASC(MID$(F$,X,1))
65 NEXT
70 PRINT D$;"BSAVE BASIC.SYSTEM,TSYS,A$2000"
80 END
```

Apple neglects to mention it in the manuals, but the ,T(ype) parameter can also be used with CAT and CATALOG to limit the display to files of a single type. For example, CAT, TBAS lists only the Applesoft programs found in the active directory. Other file types are not listed. This is helpful when looking for a particular file on a crowded disk.

Joseph Kline
APO New York

CATALOG, Tpublic.domain

I am working on a descriptive catalog of some public domain software I obtained at the North Carolina Educational Computing Conference. I am wondering if any of your readers have ever compiled or would be interested in such a catalog. I am hoping for some advice about format and detail as well as some input as to possible interest. I find it very frustrating to pick over this software to find the good and throw out the bad and duplicate. Is some moral support or feedback possible?

Janice Kay O Donovan
Oak Ridge, N.C.

The task of writing descriptive catalogs for public domain software has nearly killed many a good person. While there are a few real jewels in the public domain, you have to sift through tons of semi-precious and not-so-precious rocks to find them. It's the sifting that tends to kill people.

I myself would find a list of the "best of the public domain" interesting. I think lots of people would.

Here are two descriptive catalogs you might want to look at to see how other people do it. These two list both jewels and mundane stones, but seem to have washed away at least some of the mud.

Apple Software for Pennies, *by Bertram Gader and Manuel V. Nodar, costs $9.95 plus $1 shipping from Warner Books, PO Box 690, New York, NY 10019. It lists hundreds of public domain programs for the Apple II in the following categories: games, demonstrations and art, music and sound, utilities, education, business and home, and communications. It also has a long list of Apple User Groups and complete details on how to order public domain software from about a dozen of these groups. A highlight of the book is an 11-page instruction manual for the public domain EAMON adventure games.*

The second descriptive catalog of public domain software that I've heard of is available from the Big Red Apple Club, 1105 S 13th, #103, Norfolk, NE 68701. Big Red was started for people who don't have ready access to a local user's group. Dues are $12 a year, for which you get a monthly newsletter that describes new software added to the library. Public domain disks are sold to members for $2.50.

Compiling your own public domain catalog should be treated just like a major programming project—begin by dividing the task into small pieces and attack the pieces one at a time. Otherwise the work will swallow you whole and you'll never be heard from again. It's awful to lose subscribers that way.

Mouse trace

I have recently confirmed that a mouse cannot be used successfully as a tracing device. Let's get the word out on this before others waste energy trying. In my first attempt, I clamped the mouse to a drafting board, glued toothpicks to it, and meticulously traced over a street map of Boston.

My primitive tracing device has lately shown up in various forms, including the $50 "M.A.C.: Make Another Copy" advertised in various Apple magazines by Innovative Products of Oakbrook, IL. The M.A.C. promises a bonus; the ability to enlarge or shrink the traced artwork via a pantograph mechanism that clamps onto the mouse.

Using the M.A.C. requires practice, patience, and a desk the size of pool table. I spent three days creating this trace of a photograph of you from an old *Softalk*. Please let me know when the Thunderscan is available for the Apple II.

Brad Walters
Boston, Mass.

P.S. I would have returned the M.A.C. within 10 days for my money-back guarantee, but it took me three weeks just to assemble the darn thing.

Back when Apple first introduced the mouse I came up with an idea for a program to be called ***Paperwork*** *(no, not "works"). The core of my idea was that word processors had made the typewriter obsolete in all areas but one—filling out forms. It is nearly impossible to fill out a form with a word processor—something that was quite easy to do with a typewriter. I was convinced (and still am) that the world needed a good solution to the filling-out-forms with a computer problem. One of the really odd consequences of the computer revolution is that more forms are filled out by hand now than ten years ago.*

My plan was to have people identify the upper-left corner of a form and the relative positions of its cells by moving the mouse around the form. A few mouse movements would, in theory, recreate an image of the form on the screen and in the computer's memory. The idea crashed when I found, as you have, that the mouse is totally incompetent as a measuring device.

I recently ran across another solution to the blank form problem on AppleWorks User Group disk #28 (see page 2.33), however. Robert Merrill of Carpinteria, CA created a grid with the AppleWorks word processor that can be printed on a spare copy of the form you want to fill out. The grid shows you the line and column numbers of the cells on the form. Using the line and column information that AppleWorks displays at the bottom of the word processor screen, you can easily move the cursor to a cell's exact page position. Another possibility is to print the grid on a sheet of clear plastic and lay it over the form—this technique works even if you have only one copy of the form. Merrill suggests that you print out your file on a blank sheet of paper to make sure everything is going where you expected before you actually print on your form.

Hex calculators

Open-Apple readers may be interested in a new Casio calculator (Model CM-100). It features binary, octal, hexadecimal, and decimal conversions, arithmetic and logical shifting, left and right rotation, NOT, AND, OR, EOR, and handles numbers up to 32 bits. I got one for under $20, which I believe is around list price.

Michael Gruenthal
Durham, N.C.

Thanks for this tip. I've been using a hexadecimal calculator for several years—I can't imagine doing much in assembly language without one.

Examining high memory

April's discussion of RAM cards has renewed my interest in how to use the 16K RAM card in my Apple II-Plus.

The card was purchased some time ago and I know that the hardware is okay because Integer Basic loads and runs, and programs such as *VisiCalc* find and use the extra memory.

Thanks to your discussion, I now understand why Applesoft will never use the additional space, but I don't understand why I can't seem to get at this memory with the Monitor. When I try to set a value at any address in the upper 16K space, then list it back, I find that I have accomplished nothing.

I also have your ProntoDOS and have tried using the subroutine at $BEAF to turn the card on after running DOS-UP. The result is that the card still won't accept inputs.

I assume that I am not turning the card on properly. Can you tell me how to turn the card on from DOS 3.3? I would like to store binary data and also small assembly language programs there. If these assembly language programs cannot be called directly from Applesoft, they should be able to be linked by a routine within the 48K range which would, in turn, call the program on the language card.

Any help or hints you can provide would be much appreciated.

Ferd G. Fender
Glenview, Ill.

Begin by remembering that the Apple can only "see" 64K of memory at any one instant and that your Apple II-Plus had that much before you bought the 16K RAM card. The memory in your 16K card shares the same address range ($D000-$FFFF) as the ROMs that hold Applesoft and the Monitor.

Here is a chart of the softswitches used for turning on and off this memory area, which Apple likes to call "bank-switched memory" and old-timers like to call "language-card memory":

```
Softswitches for Apple II language card memory

                              ---$D000 to $DFFF---
                              BANK 2       BANK 1
Write-protect card
   read motherboard           R $C082      R $C08A
   read card                  R $C080      R $C088

Write-enable card
   read motherboard           RR $C081     RR $C089
   read card                  RR $C083     RR $C08B

Status (IIe/IIc only)
   high bit of $C011          1=bank 2     0=bank 1
   high bit of $C012          1=read card  0=read mbd

R=Read once    RR=read twice
```

The bank 1-bank 2 stuff refers to the fact that the addresses from $D000 to $FFFF are really only big enough for 12K of RAM. To fit in a whole 16K, the area from $D000 to $DFFF is used twice. Bank 2, by tradition, is the primary bank. The bank 1-bank 2 setting doesn't affect what you see in the area from $E000 to $FFFF, neither does it have an effect on what you see when the motherboard is turned on (ie, the switches at $C080 and $C088 do the same thing).

The R and RR stuff refers to the fact that the switches should be read, not written to, and that write-enabling the RAM card requires two reads of the respective switch, not just one.

The status stuff refers to two bytes you can read to determine the current setting of the "card" on IIes and IIcs. Your II-Plus doesn't have these switches.

Rather than try to remember all the numbers in the chart, concentrate on the two in the middle of the left column. Peeking at $C081 twice in a row lets you write to the 16K card while reading the motherboard, $C083 is the same but lets you read the card as well as write to it.

The difficult part of using the memory on the 16K card is that whenever you turn it on, Applesoft and the Monitor disappear. Consequently, you cannot examine the memory in the 16K card with the Monitor unless you move an extra copy of the Monitor onto the card.

This, however, isn't hard. Loading Integer Basic is one easy way to get a copy of the Monitor in the card. Another is with the following series of Monitor commands:

```
C081 N C081 N F800<F800.FFFFM C083 N C083
```

The $C081s fix it so you are reading from the motherboard but writing to the card. The F800<F800.FFFFM moves a copy of the Monitor from $F800 to $F800, thus giving us a Monitor to use while the RAM card is turned on. Finally, the $C083s turn the card on completely.

Now, to prove to yourself that this worked, poke a bunch of 55s at $E000 and then examine memory to see if they stuck:

```
E000.E007

E000- 00 FF FF FF 00 00 00 00
E000:55 55 55 55 55 55 55 55
```

```
E000.E007

E000- 55 55 55 55 55 55 55 55
```

They did, so this must be RAM.

To use this RAM effectively, you must approach it from assembly language. Again, you absolutely cannot get at the card from Applesoft, because when you turn the card on, Applesoft disappears and your program crashes. Your idea of linking the routines on the card to Applesoft by means of other routines in the lower 48K shows you have a basic understanding of this.

Assembly language programs will run just fine on the card, but they can't call Applesoft routines. They can't use Monitor routines, either, unless you move a copy of the Monitor over to the card.

The DOS-UP hook at $BEAF does in fact turn on the card (bank 2) for both reading and writing. You can't use this hook from the Monitor, however. Every time you type a character, DOS-UP has to turn the language card on twice—once so DOS can examine the character as you type it and once so DOS can examine the character as the Monitor prints it to the screen. Just typing BEAFG and return causes $BEAF to be called (and the card to go on and off) twelve times. The routine at $BEAF will dutifully turn it on a thirteenth time for you, causing the Monitor to disappear. The RTS at the end of $BEAF that should return you to the Monitor sends you scurrying through DOS instead. Usually you'll hit a break before your disk drive goes up in flames, but that isn't guaranteed. $BEAF is sold as-is; it is useful only from assembly language.

PR#3 and ProDOS

In plain, easy to read assembly code, how does one properly activate the 80-column card when using ProDOS on a IIe (i.e., without using a form of PRINT CHR$(4);"PR#3", but by direct manipulation).

William M. Reed
New Orleans, La.

It all depends. Are you writing an assembly language program that will be in a binary file and coexist with Basic.system, or are you writing one that will be in a system file and will manipulate the ProDOS kernel's machine language interface?

It's a critical difference. If you're writing a system program, you have to handle and keep track of all peripheral connections—screen, printer, and so on—yourself. All the ProDOS kernel was designed to help you with is mass storage (disk) devices.

In this case, in pure, unadulterated machine language, the answer is to do just a touch more than the obvious:

```
A9 99     LDA #$99      get a harmless ASCII chr
20 00 C3  JSR $C300     cold start the card
```

This assumes that you have checked to make sure there's a card to jump to first—use the ProDOS MACHID byte at $BF98; bit 1 (7654 3210) will be "1" if ProDOS recognized an 80-column card during the boot process. It's also very important that you put something harmless in the A register before calling $C300—$99 is a control-Y, the code for homing the cursor without clearing the screen. Dennis spent a whole afternoon once just figuring out that he was trying to initialize the card with a leftover $15 in the A register. This is control-U, the code for deactivating the 80-column firmware; you definitely can't turn on 80-columns by printing that character.

Now, on the other hand, if your assembly language program runs under Basic.system, there are two ways to go. One is to simply poke $C300 into VECTOUT (vector out) in the Basic.sytem global page at $BE30 and then print something, like this:

```
A9 00     LDA #0
8D 30 BE  STA VECTOUT     poke $BE30,$C300
A9 C3     LDA #$C3
8D 31 BE  STA VECTOUT+1
A9 99     LDA #$99        get a control-Y
20 ED FD  JSR COUT        print it via $FDED
```

The other way to go is to poke the ASCII string PR#3, followed by a return, into the keyboard input buffer at $200, and then call DOSCMD in the Basic.system global page at $BE03. According to Apple's ProDOS Technical Note #2, this trick works for all Basic.system commands except dash, RUN, LOAD, CHAIN, READ, WRITE, APPEND, and EXEC (the main ones you wanted to use, right?).

The common technique under DOS 3.3 of printing command strings such as "control-D PR#3" directly from assembly language doesn't work under Basic.system.

If you use the keyboard input buffer-DOSCMD technique, you should check the carry for an error when Basic.system returns to you. If an error has occurred, you can do one of three things. A JSR ERROUT ($BE09) will send control to your Applesoft ONERR routine. A JSR PRINTERR ($BE0C) will print a suitable error-message on the screen and return to you. The third possibility is to handle the error yourself completely. In this case, make sure you clear the carry before RTSing back to Applesoft.

Echo overwrites program

I am using a IIe with Extended 80-Column Card and Epson MX-80 printer. The printer is connected through a Dumpling GX in slot 1.

Occasionally, when using conditional branching (mainly IF...THEN), I get an UNDEF'D STATEMENT ERROR and my program listing is garbled.

After trial and error I have found the program will run normally when I change the printer interface card to a Prometheus PRT-1. All switches on the Dumpling GX are set as recommended for my printer. Can you suggest a solution so that I can use the Dumpling GX?

Gene Watson
Sulphur Springs, TX

Here's a wild guess. I suspect the Dumpling firmware that echos printer characters on your display screen expects you to use 40-column mode. If you print with the echo in 80-column mode, subtle bugs may cause the echoed characters to miss the screen and overwrite the beginning of your program instead.

There are two solutions. One is to turn off 80-column mode before you try to print. This is Apple's recommended solution to the problem (see "Errata to the Apple II 80-Column Text Card Manual," page 4). Obviously programs such as AppleWorks and **Apple Writer** *don't follow Apple's advice in this matter, however, so why should you?*

The other solution is to use the proper control codes to tell the Dumpling not to echo printed characters to the screen. I don't have the Dumpling control codes handy, but for most cards the command sequence is something like "control-I 80N". The eighty tells the card how often to add carriage returns—if your card has a code for "never" (zero on many cards), it's usually your best choice. For more, see our November issue, pages 84 through 86.

A final possiblity is that the manufacturer of the Dumpling can provide you with updated ROMs for the card that will solve the problem. Giving the company a call can't hurt.

Hard drives and *Apple Writer*

Do you know how to put *AppleWriter IIe* on the Sider hard disk system?

Louis A. Marinaccio
Bel Air, Md.

Good old DOS 3.3-based **AppleWriter IIe** *uses a customized version of DOS and is copy-protected. Consequently, it is difficult to use from a Sider or RAMdisk.*

However, there is a secret provision within **Apple Writer IIe** *that allows it to work with a Corvus hard drive, but only if the hard drive is in slot 6 and the floppy in slot 7 (backwards from most Sider installations). Press "C" while the* **Apple Writer IIe** *disk is booting and a prompt will appear asking if you want to enable the Corvus. Paul Lutus,* **Apple Writer***'s author, revealed this tidbit in a letter to* ***Byte*** *(August 1983, page 32). Lutus seems to like this sort of undocumented feature; a serial interface driver for the game connector was secreted within the original version of* **Apple Writer**.

The newer ProDOS version of **Apple Writer** *is unprotected and can simply be copied to a ProDOS volume on any RAM or hard disk. To get it up and running, you must first set the prefix to the subdirectory the program is in, then execute AW.SYSTEM. If you are using a RAMdisk that appears to be in slot three, however,* **Apple Writer** *will disconnect it. The simpliest way to avoid this is to move the RAMdisk slot assignment somewhere else. If you are using an auxiliary-slot RAMdisk, you'll also need to make sure the RAMdisk driver doesn't try to use the auxiliary 64K memory bank, because* **Apple Writer** *also uses it.*

ProntoDOS on the Sider

How can I get *ProntoDOS* onto the Sider? In particular, I would like to speed up floppy disk access, get the free sector list at the top of the catalog, and make sure that a disk initialization doesn't leave a copy of DOS or a HELLO program on a floppy.

David Holladay
Madison, Wisc.

There are two primary ways to go about using ***ProntoDOS*** *and the Sider. One is to actually put* ***ProntoDOS*** *on the Sider's boot track, so that it comes up when you turn on the system. The other is to leave DOS 3.3 on the boot track, but modify it into* ***ProntoDOS*** *after the system is started.*

To get ***ProntoDOS*** *on the Sider's boot track, begin by booting the* ***ProntoDOS*** *disk, then use the program PRONTO UPDATE to make the DOS enhancements you want (such as the free-sector and initialization items you mention). You should now have an image of DOS in memory that is the one you want to come up when the Sider boots.*

Now insert the Sider DOS 3.3 utilities disk and RUN MAKE BOOT TRACK. The manual says the MAKE BOOT TRACK program is to be used when you move the Sider controller card from one slot to another, however, it can also be used to simply place an image of the DOS in memory onto the Sider. Don't be surprised, however, when it asks you which slot you intend to move the controller card to; just answer with the slot number it is already in.

You can skip MAKE BOOT TRACK if you have not yet initialized your Sider. Insert the Sider installation disk and RUN HELLO (do not boot the disk) after booting and enhancing ***ProntoDOS****. During the initialization process the image of* ***ProntoDOS*** *in memory will be placed on the disk.*

If you have some DOS 3.3 programs that don't work with ***ProntoDOS*** *and would prefer to have virgin DOS 3.3 on the Sider, another alternative is to move the program HELLO PRONTO-DOS from the* ***ProntoDOS*** *disk onto the Sider and run it, after booting the Sider, whenever you want the DOS in memory changed into* ***ProntoDOS****.*

The two disadvantages of this technique are that there's no way to install the enhancements you want and sometimes it doesn't seem to work. The DOS 3.3 image on one version of the Sider installation utilities (distributed between about October 1985 and March 1986) accidentally had three mixed-up bytes in one of the areas HELLO PRONTO-DOS uses to identify DOS. If your Sider boots with this DOS image, HELLO PRONTO-DOS will tell you it doesn't recognize the DOS in memory and will refuse to execute. The solution is to use MAKE BOOT TRACK to put a clean copy of DOS 3.3 onto your Sider.

Incidentally, ***ProntoDOS*** *speeds up not only floppy disks, but hard disks and RAMdisks as well. See February 1985, page 10, for the incredible details.*

Open-Apple

[redacted]

is written, edited, published, and

Business Consultant Richard Barger
Technical Consultant Dennis Doms
Circulation Manager Sally Tally

Open-Apple
P.O. Box 7651
Overland Park, Kansas 66207 U.S.A.

ISSN 0885-4017. Published monthly since January 1985. World-wide prices (in U.S. dollars; airmail delivery included at no additional charge): $24 for 1 year; $44 for 2 years; $60 for 3 years. All back issues are currently available for $2 each; a bound, indexed edition of Volume 1 is $14.95. Index mailed with the February issue. Please send all subscription-related correspondence to:

Open-Apple
P.O. Box 6331
Syracuse, N.Y. 13217 U.S.A.

Subscribers in Australia and New Zealand should send subscription correspondence to ***Open-Apple***, c/o Cybernetic Research Ltd, 576 Malvern Road, Prahran, Vic. 3181, AUSTRALIA.

Open-Apple is available on disk for speech synthesizer users from Speech Enterprises, P.O. Box 7986, Houston, Texas 77270 (713-461-1666).

Unlike most commerical software, ***Open-Apple*** is sold in an unprotected format for your convenience. You are encourgaged to make back-up archival copies or easy-to-read enlarged copies for your own use without charge. You may also copy ***Open-Apple*** for distribution to others. The distribution fee is 15 cents per page per copy distributed.

WARRANTY AND LIMITATION OF LIABILITY. I warrant that most of the information in ***Open-Apple*** is useful and correct, although drivel and mistakes are included from time to time, usually unintentionally. Unsatisified subscribers may return issues within 180 days of delivery for a full refund. Please include a note from your parents or children confirming that all archival copies have been destroyed. The unfullilled portion of any paid subscription will be refunded on request. MY LIABILTY FOR ERRORS AND OMISSIONS IS LIMITED TO THIS PUBLICATION'S PURCHASE PRICE. In no case shall I or my contributors be liable for any incidental or consequential damages, nor for any damages in excess of the fees paid by a subscriber.

Open-Apple is neither affiliated with nor responsible for the debts of Apple Computer, Inc.; "tinaja questing" is a trademark of Don Lancaster.

Source Mail: TCF238 CompuServe: 70120,202

More Sider secrets

After several more Sider crashes similar to the disaster discussed in August (page 63) and October 1985 (page 79), I finally have an answer to the "hard disk life" I've been experiencing.

After each new crash I would be fenced out of additional areas of the hard disk, losing files each time. I would have to reinitialize the Sider to put things right. The crashes occurred during disk accesses and it appears that my original association of the crash with pushing reset while *Apple Writer* was printing was just coincidental. Each time I reinitialized the Sider there was no indication of a problem.

The breakthrough came when I began to work with a new 20 meg Sider that I daisy-chained to the first as /HARD3 and /HARD4. The new utilities package that came with the 20 meg unit included Apple's ProDOS BACKUP program. BACKUP would crash when creating the /HARD1 catalog for the backup disks. After each crash I would have to cold start to regain control of the Apple. The crashes weren't BACKUP's fault, but resulted from the program attempting to access every file on /HARD1. When BACKUP began to do business with a poisoned section of the disk, the Sider would enter hyperspace and BACKUP would be left hanging.

The next time I reinitialized the Sider I got a "non-media error 95" during final verification. This was the first time I had seen this message. The Sider would apparently function normally after reinitialization until a file happened to hit on the track with the problem. Once this track was accessed the whole process of gradual bad-blocking of more and more of the Sider would begin anew.

I called the Sider hotline to find out what a "non-media error 95" was. The very-helpful technician I talked with was not sure of the significance of this message, but suggested that it probably indicated a marginal track. He suggested that I use some undocumented features of INSTALL PT#4 (on the DOS 3.3 utilities disk) to run a system check.

To access these utilities, BRUN INSTALL PT#4, then hit the R key within one or two seconds. This results in a menu-driven package of marvelous machinery. I don't know why they keep these utilities a secret—there are some really useful tools in this program—perhaps because there is plenty of potential to wreak havoc on your disk if these tools are misused. Among other processes, you can run a non-destructive check of all tracks—apparently this is the same routine that writes dots to the screen as it verifies the disk after initialization.

This check gave me a "non-media error 95" again. An option during this check is to reassign one of a couple of dozen spare tracks to replace any marginal track. This option carries the obvious caveat that reassignment may destroy data on the disk. I was able to reassign the offending track, 03313, without a loss of data. Apparently I had not written to this track since last reinitializing the disk. My old Sider has performed flawlessly since the replacement. I suspect that this track had been marginal since I first began to use the disk.

It looks as if any time the Sider is not able to complete a disk function because of a flaky track it can lose all sense of responsibility and viciously tear away at previously uncorrupted files—at least that was my experience.

Some of the utilities hidden on INSTALL PT#4 include provision for reformatting just the DOS 3.3 portion of the disk without touching areas allotted to other operating systems. Also included is a provision for zeroing the catalogs for just the ProDOS portion, leaving the other sections alone.

Apple's BACKUP program is great. It works so well that backing up the entire disk is a piece of cake. According the tech I spoke with a new backup utility for the Sider's DOS 3.3 section, which will work similarly, will be available soon. Their present DOS 3.3 utility is a loser because it prompts you for each file. You have to stay alert during the entire backup procedure and it's painfully slow. The ProDOS BACKUP program does almost everything for you but put the floppies in the drive and it won't let you make a mistake.

The new utilities supplied with the Sider seem to place more emphasis on parking the heads before turning the unit off. I asked the tech about this, since they had told me earlier that parking the heads was only important before moving the unit around. His response was that there had been some kind of problem related to the Apple writing to the hard disk when it wasn't up to speed if the heads were positioned over prime real estate on the disk. It seems that parking the heads is more of a precaution against this eventuality than against a head crash.

Do you have any more information on the upgrade for *Apple Writer 2.0* you wrote about in January (page 97)? The dealer where I bought my original doesn't now anything about an upgrade and if there are any improvements I'd sure like to have them even if they aren't significant.

Donald Beaty
San Mateo, Calif.

Thanks for all the Sider secrets. As of the end of March, when Dennis last talked a list of available updates out of Apple's district sales office here in KC, "customers having printing problems with ***Apple Writer 2.0*** *and third-party interface cards may take the disk to the seller, who will correct the fault through an update utility." This is what you're looking for, but Apple has never clarified what you are supposed to do when the dealer doesn't know what he's supposed to have.*

Other Apple II updates on the list are:

```
Apple IIe enhancement
  $70 from dealer (installed). Self-installation
    recommended so you can keep your old chips.
AppleWorks 1.3
  $20, get mailer from dealer
ProDOS 1.1.1
  Free update from dealer
SuperPilot Log
  Free update from Apple Customer Relations MS 27-F
Pascal Version 1.3
  $125 + original APPLE1 disk (no personal checks)
     Apple Computer Pascal 1.3 Upgrade
     P.O. Box 306
     Half Moon Bay, CA 94019
Logo 64K
  Disks that don't work on IIc because of spiral
    protection scheme can be replaced through
    Apple's Media Exchange Program. Upgrade to
    Logo II (128K) not available.
ProDOS Users Disk
  If MouseText characters appear, disk can be
    replaced through Apple's Media Exchange
    Program.
```

Apple's Media Exchange Program, incidentally, works like this. If you have a damaged Apple disk or manual, you get an "Exchange Program Card" from your dealer. Fill it out and send it to Apple along with the damaged disk or manual. Apple will send you a replacement at no cost within 3 to 4 weeks.