Open-Apple™

September 1986
*Vol. 2, No. 8**
*Last month's issue was really No. 7

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

Releasing the power to everyone.

Missing No. 7 found

Bugs in software are forgivable, but the idiot who numbers our issues (me) may never be forgiven. Last month's issue was really number 7, not number 8. We have a greater appreciation now for how many of you save our back issues—we've been buried by requests for issue 7. Please help me recover from this mistake by marking it as Vol. 2, No. 7 right now.

Technical tools liberated

Aftershocks from Apple's corporate reorganization more than a year ago were felt again during August when Apple and A.P.P.L.E. Co-op (the world-wide users group based in Seattle that publishes ***Call -A.P.P.L.E.*** magazine), jointly announced the formation of the Apple Programmers and Developers Association, or APDA.

This new association, which is open to anyone who's interested, will be a central source for Apple development tools, including Apple's own technical notes, manuals, and development software, as well materials from a variety of third-party companies.

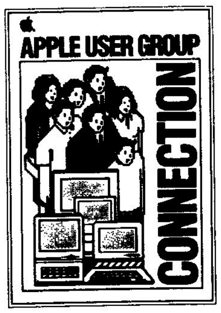
Association members will receive a quarterly catalog and newsletter, regular update bulletins, and ordering privileges. The association will support both Apple II and Macintosh programmers.

To join, send $20 for first year dues to the Apple Programmers and Developers Association, 290 SW 43rd St, Renton, WA 98055. For more information you can call 206-251-6548.

If this association fulfills its promise, it will be a major factor in the continued success of the Apple II. To quote from the seventeen-month-old May 1985 issue of ***Open-Apple*** (page 34):

Apple appears to have enough ***money*** *to break through the hardware limitations of the Apple II and appears... headed in that direction. But the major question at this juncture is whether Apple has enough* ***sense*** *to actively and purposefully "expose the inner flesh" of these machines to interested members of the general public. That is the factor that has been shown to determine the ultimate success of any personal computer, and that includes present and future models of the venerable Apple II.*

It certainly impresses me to see an institution as large as Apple develop so much sense so quickly. Now that the developer's association is here, look for a new Apple II any minute.



Another result of Apple's reorganization is that it's keeping its promise of better support for user groups. A new logo (left) has been created that groups can use on their newsletters, memberships cards, letterheads, and promotional materials. An 800 number (800-538-9696, ext 500) has been established that U.S. residents can call to find their nearest group. Apple is sending groups a regular monthly mailing and participates in a special user-group Forum that is part of MAUG (the Micronetworked Apple Users Group) on the CompuServe Information Service.

Apple's head of developer technical support is swapping subscriptions to the Apple II or Macintosh technical notes for subscriptions to user group newsletters. Apple's mailing list people are taking steps to ensure that all members of user groups are included in future mailings from Apple. Apple's office of special education is polling user groups to determine whether it can facilitate the exchange of information on the computer-related needs of disabled children and adults.

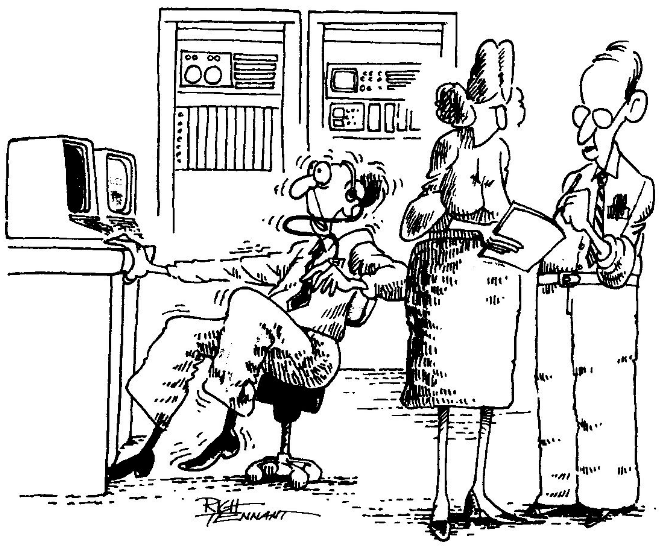
Call Apple's 800-number today to make sure your group has made connections with Apple. If you don't belong to a user group, now is a good time to make the connection.

Yet another positive aftershock of Apple's reorganization was the recent promotion of Del Yocam to Apple's Chief Operating Officer. Yocam will manage not only Apple's product development, manufacturing, and distribution operations, as before, but also sales and marketing. Yocam's promotion is expected to allow John Sculley, Apple's chairman and president, to focus on future directions, strategic issues, and business development opportunities. "My time will be focused on building Apple and Del's on running Apple," Sculley told ***Business Week***.

Previously, Yocam was head of the Apple II group, which developed, engineered, manufactured, and marketed all Apple II products. Yocam, whose wife taught elementary school for eleven years and who himself taught night classes for seven, is known for his strong support of Apple's efforts in the education market. Yocam is interviewed in the August/September issue of ***II Computing*** ($11.97/yr—6 issues, P.O. Box 1922, Marion, OH 43306 614-383-3141).

All is not perfect in the Apple world, however. Many ***Open-Apple*** subscribers in rural areas have found themselves without a local supplier in the wake of Apple's recent reduction of authorized dealers. In this case Apple's purpose seems to be in the right place—it wants to develop dealers who will concentrate on specific markets, learn them inside out, and support them with all their hearts.

Rather than keeping the dealers whose salespeople knew AppleWorks from *Apple Writer*, however, Apple seems to have kept the ones who had the highest sales volumes. A dealer who knows the Apple product line and who



"I DON'T THINK OUR NEWEST NETWORK CONFIGURATION IS GOING TO WORK. ALL OUR TRANSMISSIONS FROM CLEVELAND ARE COMING IN OVER ERNIE'S WALKMAN."

can demonstrate a half-dozen software packages is a precious and valuable thing. Apple knows that and says it wants all of its dealers to be like that, yet every couple of years we see a bloody dealer purge based on other standards.

Apple painted itself into a corner several years ago when it cancelled its agreements with all dealers who were selling by mail order. The real reason it did this was to appease dealers who were mad about some big mail order houses that were selling high volumes of machines at low prices. However, to appease end-users mad about losing a low-cost source of machines, Apple said it was cutting off mail order dealers because they didn't provide high-quality technical support.

This is ludicrous. The level of technical support you get from a dealership has nothing to do with whether you contact it in person, by phone, or by mail.

For example, the best dealer you'll find for portable Apple IIcs is Roger Coats (P.O. Box 171466, San Diego, CA 92117 619-274-1253). Roger specializes in and stocks for immediate shipment C-Vue LCD displays, carrying cases, battery power systems, cigarette lighter adapters, and IIc modems, clocks, and memory boards. There is no one at Apple itself or at any other dealership in the country with as much information on the products that are available for making a IIc truly portable.

This kind of support for specialized markets is only possible from dealers who consider their marketplace to be country- or world-wide and who, as a consequence, deal with customers by phone and mail.

Likewise, so few store-front dealerships have been able to provide decent support for people interested in programming that, as mentioned earlier, Apple has had to form a "strategic alliance" with a user group to provide this support by ... shhhh ... mail order.

More, not fewer, specialized mail order dealers are badly needed. Many ***Open-Apple*** subscribers, for example, would appreciate a dealer specializing in statistical analysis software. Imagine being able to make one call to find out the cheapest way to outfit a 16-computer lab with software for a class, or to find out if a specific package works with unequal N and what it does about missing data. Could such a dealer possibly operate any other way than by phone and mail?

The rural areas of the world already have enough problems without having to worry about where they're going to buy their next Apple II. Apple has every right to insist that its dealers support its products—but when is it going to wake up and select its dealers on that basis, rather than on easy-to-measure but mindless criteria such as method or volume of sales?

Apple Assembly Line **($18/yr, P.O. Box 280300, Dallas TX 75228 214-324-2050) has come up with a great example of using sound to analyze data**, a topic discussed here briefly in March (page 2.9). In AAL's July issue, Bob Sander-Cederlof uses short tone-through-the-speaker subroutines to determine how many "machine cycles" certain 65816 machine language instructions take to execute. Of course, this kind of information is in the standard reference manuals for the chip, but it turns out that for some of the instructions some of the standard references are wrong.

Determining how many cycles an instruction takes can be tedious work. The typical method is to write a loop that will run millions of times. Then you use a stopwatch to figure out how long the loop takes both with and without the opcode under examination. Subtract the shorter time from the longer and divide by the number of loops. Multiply by a conversion factor that changes millionths of a second into machine cycles and you've got it.

Instead of this rigamarole Bob simply created a continuous tone-loop that included an opcode whose number of execution cycles he was certain of. When he pressed a key, control passed to a duplicate loop that included the opcode under examination instead. If the pitch of the tone changed, the number of cycles used by the new opcode was different (higher pitch meant fewer cycles).

Using this simple program, Bob figured out that the branch-long (BRL) instruction takes four cycles, rather than the three cycles claimed on the manufacturers' data sheets; and that the branch-always (BRA) instruction takes three or four cycles (depending on mode and page crossings), rather than the two claimed by the manufacturers. Don't overlook the power of sound in your own programs.

When I keep mentioning the same software package over and over here in ***Open-Apple***, it's because it's good. Glen Bredon, author of the Merlin assembler, (521 State Road, Princeton, NJ 08540) just keeps adding features to his $40 PROSEL package. If you have a hard disk, or if you are using a 3.5 inch disk in conjunction with a RAMdisk, this package is invaluable.

As mentioned before, the basic package is a program selector that runs automatically whenever you quit a ProDOS program. Leave AppleWorks, for example, and your own menu of available software instantly appears on the screen—no prefix or pathname required.

The program selector has become only a small part of the software on this disk, however. In addition there's a program for backing up and restoring ProDOS devices (including the ability to backup the contents of a RAMdisk into a *file* that can be automatically reloaded into the RAMdisk on startup); a RAMdisk driver for auxiliary slot cards; a program for copying, deleting, and otherwise maintaining files (including a directory sorter that works); a fast disk copy program; a disk zap program; a simple password protection program; a program for putting DOS 3.3 into a ProDOS file; UNO.DOS, which allows 3.5 inch UniDisks to be used as 140K DOS 3.3 disks (with the rest of the disk left for ProDOS); a patch for *Pinpoint* that allows it to be used with the program selector; BEACH.COMBER, a program that rearranges a hard disk so all directories and files lie in contiguous areas, which speeds up file access; MR.FIXIT, a program that checks out all the directories on a ProDOS volume, looking for more than a dozen kinds of errors, and fixing those it finds; and, as they say, more.

If you've been neglecting to enhance your IIe, or if you sometimes wish you hadn't, you may be interested in a device called Switchback ($59.95 + 2.50 shipping, Computer Accents, P.O. Box 5905, Kingwood, TX 77325). This is a small board with six sockets, four for the old and new Monitor ROMs and two for the old and new video ROMs, which plugs into your motherboard. It has a built-in switch that determines whether your computer normally starts up with the old or new ROMs and it has an external switch you can press while power is applied to start up the alternate set.

I suspect the device will be of particular interest to software developers who want to test their work on both original and enhanced IIes, but who would like to avoid having an extra machine around. That's why I bought one, anyhow. It works as advertised.

A few notes: you do have to turn the computer off to switch from one ROM set to the other. You do need both sets of chips—they are not supplied with Switchback (you kept your old ROMs as recommended in ***Open-Apple*** May 1985, page 38, right?). Switchback isn't compatible with European-type motherboards that have the auxiliary slot in line with slot 3, nor with any of the motherboard-based 65816 or Z-80 cards we've tried. There is also a potential conflict with any long card in slot 5, 6, or 7 that has a "support foot" or that otherwise protrudes below the normal height of a card.

A little over a year ago Uncle DOS got a question about how to transfer Apple II graphics to T-shirts. Not knowing the answer, and being pretty sure he never would, he filed the letter in the circular file. Now, oh gentle questioner, we sure hope you renewed your subscription because we have an answer.

What you need is an Underware Ribbon from Diversions (505 W Olive Ave, Suite 520, Sunnyvale, CA 94086 408-245-7575). These ribbons are available for most dot matrix printers. You print your graphic on a piece of erasable bond paper using the Underware Ribbon, then you transfer the graphic from the paper to cloth (50-50 cotton-polyester fabric recommended) with an iron. The instructions recommend using an image no larger than twice the size of your iron.

Prices vary by what kind of printer you have, but for the Imagewriter, for example, black ribbons are $14.95, single color ribbons are $16.95, a multi-color ribbon for the Imagewriter II is $24.95, and a black ribbon with 5 pens of various colors (to be used with manual color-fill subroutines) is $24.95. A single ribbon is good for 30 to 100 images, depending on how much ink is used by each image.

The Academic Courseware Exchange (4141 State St., Santa Barbara, CA 93110 800-235-6919, 800-292-6640 in CA) has just published a free catalog of programs for use in college and university courses. The purpose of the Academic Courseware Exchange is to provide a means whereby students can obtain course software just as they now do textbooks.

Faculty members who decide they'd like to use a program in their class are supposed to make arrangments with a local Kinko's photocopy center to order the software. Students then go to Kinko's to purchase the software, which is priced between $7 and $30 per package. If there's no Kinko's store near a particular school, Kinko's headquarters will arrange to deliver the software through a local bookstore.

Faculty members who have developed courseware they'd like to see listed in the catalog should write or call the courseware exchange and ask for a free developer's handbook. Kinko's handles production, distribution, and marketing of the software and pays a royalty based on sales. Any subscribers who develop for or buy from the exchange are encouraged to let Uncle DOS know what they think of it.

Ask (or tell) Uncle DOS

Dual custom printers

There's more than one way to set up more than one custom printer in AppleWorks. What I did was make a duplicate of the 9-block SEG.PR file on the AppleWorks disk. I named the two files SEG.PRINTER1 and SEG.PRINTER2. Then I wrote an Applesoft startup program that asks which printer I want to use. The appropriate file is renamed SEG.PR and AppleWorks is loaded. To switch printers you have to exit AppleWorks, but that doesn't bother me. Here's my startup program:

```
3000 REM  Multi-Custom Printer Startup Program

4000 TEXT : HOME : NORMAL : D$=CHR$(4)

4100 REM  If present, rename SEG.PR
4110 ONERR GOTO 4130 : REM duplicate filename trap
4120 PRINT D$;"RENAME SEG.PR, SEG.PRINTER.1"
4130 ONERR GOTO 4150
4140 PRINT D$;"RENAME SEG.PR, SEG.PRINTER.2"
4150 POKE 216,0 : REM cancel ONERR

4200 PRINT "PRINTER #1 OR #2? "; :
     GET A$ : PRINT A$
4210 IF A$ < > "1" AND A$ < > "2" THEN END
4220 IF A$="1" THEN
     PRINT D$;"RENAME SEG.PRINTER1, SEG.PR"
4230 IF A$="2" THEN
     PRINT D$;"RENAME SEG.PRINTER2, SEG.PR"

4240 ONERR GOTO 4260
4250 PRINT D$;"-APLWORKS.SYSTEM" : END
4260 PRINT D$;"-MACRO.SYSTEM" : REM for MacroWorks
```

Bert Kersey
Beagle Bros
San Diego, Calif.

Mail label musings

I haven't been able to find a software program to use with my computer that will work with pressure sensitive mailing labels. The programs I have seen all print out in a single column of labels (only one label wide). Using such a program will require the purchase of a special tractor modification for my printer. Please advise me on how to proceed with this problem.

T. A. Dumetz
Los Angeles, Calif.

You bring up an interesting problem. Standard one-wide pressure sensitive labels can't be used with several of the printers commonly found hooked to Apple IIs. On the Apple Imagewriter, for example, the tractor wheels can't be moved close enough together to accommodate standard labels.

Imagewriter owners can easily solve their problem, however, with extra-wide one-up labels. For example, Moore Business Products (P.O. Box 20, Wheeling, Ill. 60090 800-323-6230) stocks one-up pressure sensitive labels in seven different widths from 2.5 inches to 7.75 inches. The narrowest ones that will work on an Imagewriter are 4 inches wide. We've used literally thousands of them here at **Open-Apple.**

Some other printers, including many Epsons, have tractors that are only barely (or not at all) adjustable. A possibility in this situation is a program called **Muse Address Book,** *which will print labels from 1 to 6 across. Muse software entered bankruptcy in September 1985, but its assets were eventually purchased by another company, which is now selling and supporting Muse's products. The retail price of the* **Muse Address Book** *is $49.95, but a hefty discount is available if you call or write Muse Software, Damascus Centre, Damascus, Md. 20872 301-253-3553. (Thanks to subscriber Doug Brower in Oshkosh, Wisc. for Muse's new address.)*

One important thing we have learned about pressure sensitive labels here at **Open-Apple**, *incidentally, is to never roll them backwards through your printer. When you finish using them, tear off the unused labels behind the printer and roll the remaining ones forward through the printer. This wastes a few labels but is far better than having a label get stuck underneath your printer's platen. This often happens when rolling the labels backward and usually requires taking the printer apart to get the wayward label unstuck.*

Disk resurrection

Is there a way to bring back an AppleWorks data disk that has crashed? I did not update my backup disk and the information that is on this disk is extremely important to me. I would appreciate any assistance you could offer.

Melvin Katz
Davie, Fla.

Ah, yes, the old disk-crash problem.

First, examine the troublesome disk for physical damage. Will the magnetic media rotate freely inside the cover? Is the cover crimped or heat damaged? Was something spilled on the disk? Problems such as these can sometimes be solved by cutting open the disk cover on both the damaged disk and on a good disk. Carefully remove the magnetic media from both disks. Clean the damaged media (if necessary) and place it in the undamaged cover. It may now be readable. If so, make a couple of copies of it and throw the damaged material away.

If the disk's problem isn't physical damage, a likely problem is bad centering inside the disk drive. Start up AppleWorks and tell it you want to add a file to the desktop. Before telling it where the file is, open the door on the disk drive that holds the damaged disk. Then tell AppleWorks which disk the file is on. The disk drive light will come on. At that point close the drive door. Closing the drive door while the disk is spinning is the best way to get the disk properly centered.

If your lost file still won't load, try putting the disk in a different drive and repeat the procedure. When disk drives start to get out of alignment, the primary symptom is that disks written on one disk drive can't be read on another. Make sure you try to read the troublesome disk using all of your various disk drives.

If you have still have had no luck reading the disk, you should at least know by now whether the damage lies in the disk's directory or in the file itself. If AppleWorks will show you a list of the files on the disk but coughs while loading the one file you really want, the file is bad. If AppleWorks won't even show you the list of the files on the disk, the directory is bad.

In either case, the next step is to ***put a write-protect tab*** *on the bad disk and try to make a copy of it. Start with the COPYA program from the DOS 3.3 System Master disk if you have it. Boot DOS 3.3 and hand-enter POKE 47426,24. This defeats some DOS 3.3 error-checking. Then RUN COPYA and attempt to copy the bad disk. If the copy succeeds, startup AppleWorks and try to load your files from the copy. This trick will sometimes recover disks that have been through a drive that was slightly off-speed.*

If you've gotten this far without success, it's time to buy yourself a copy of **Bag of Tricks 2** *($49.95 plus shipping from Quality Software, 21610 Lassen, #7, Chatsworth, CA 91311, 818-709-1721).* **Bag of Tricks 2** *is a set of four programs designed specifically to help you recover damaged disks. First concentrate on the program INIT—use it to make a copy of your damaged disk.*

If your problem is a damaged file (rather than a damaged directory), try loading the file off of the new disk that INIT creates. If the problem is a damaged directory, on the other hand, use the program FIXCAT to massage INIT's copy. FIXCAT will scan the disk and attempt, often successfully, to rebuild the bad directory.

If you get this far and still have had no luck, you're in pretty deep.

Dennis and I have tossed around the idea of offering a data-disk recovery service (NOT to include copy-protected program disks) for a fee—say maybe $30. We have the needed hardware, software, and expertise—all we need is the time. Is such a service needed? How much is it worth? Is there already somebody out there doing this that we don't know about?

DIFiculties revisited

I read with much interest and empathy Douglas Sietsema's August letter (page 2.56) about the difficulty of moving a *VisiFile* data base to AppleWorks with DIF files. Having spent many long hours trying to do almost exactly the same and a few other things, I really felt sorry for him.

Not only did I finally accomplish a *VisiFile* to AppleWorks transfer—about a year later I transferred the data from AppleWorks to a Macintosh program called *Overview*. In each case I eventually determined that at least part of the problem was that the DIF file had to be converted from column format to row format.

Having some experience with *Advanced VisiCalc*, I knew that it allows a choice of saving DIF files in either a row or column format. That was the key. I read the *VisiFile* DIF file into *Advanced VisiCalc* using a column format. Then I resaved it using a row format. Then I converted the file to ProDOS and AppleWorks read the file correctly. Going from AppleWorks to the Macintosh again required a side trip through DOS 3.3 and *Advanced VisiCalc*.

David G. Story
Sault Ste. Marie, Ontario

I suspect your side trips through **Advanced VisiCalc** *did more than simply convert your DIF file from column format to row format. It also got rid of the additional header items I mentioned last month. I also suspect you could use just about any spreadsheet that supports DIF files (with the exception of AppleWorks) to clean up and reformat your DIF files.*

As you point out, the AppleWorks database gives you no opportunity to specify row or column format *when loading or saving DIF files. However, the AppleWorks spreadsheet does present you with a choice when printing (saving) DIF files.*

*If the only problem with a DIF file was that its row-column format had to be reversed, you could do it by passing the file through the AppleWorks spreadsheet as well as through **Advanced VisiCalc**.*

*Incidentally, to get a file to appear on the multi-record database screen in the same order that it appears on the spreadsheet (records in rows, categories in columns), you must use the **column** format when printing (saving) the DIF file. This is kind of odd, since the default is the row format. Limited testing here indicates that if you have more than 30 categories in a DIF file the AppleWorks database simply lops off the extra ones—it doesn't refuse to load the file.*

Cartoon reprints

How can I obtain the rights to reprint a cartoon that appeared in your newsletter?

L. J. Lynch
Fort Worth, Texas

*All of the **Open-Apple** cartoons published to date have come from the same artist. Contact him directly for full information—he charges by circulation size and has very reasonable rates for small publications. He is Rich Tennant, 93 Forest Hills, Apt II, Jamaica Plain, Mass. 02130 617-522-0821.*

***Open-Apple** welcomes submissions of Apple II-related material from any cartoonist.*

DOS 3.3 text file length

I had the problem recently of having to combine about a hundred different sequential text files into one compact file. Each small file consisted of a combination of numbers and strings. I ended up with the inelegant solution of converting each small file to all strings, adding up the LEN of the strings, throwing in a dummy string, and saving all that to the main file. Then I'd save the next file using the location of the dummy string (i.e. WRITE FILE, Bxxx; where xxx is the byte number of the last dummy string).

With a few fits and starts it eventually worked, but the question I want to ask eluded me in any research I tried to do. Namely, operating under DOS 3.3, how can you ascertain the number of bytes in a text file? Isn't there some nice convenient PEEK just waiting for me to learn its address?

Jim Menick
Peekskill, N.Y.

There is no magic PEEK to determine a text file's length under DOS 3.3. DOS simply doesn't know how many bytes are in a text file. If you must know, the only way to find out is built a GET loop and count the characters until you hit an END OF DATA error.

*In your case, it might have been easier to use the DOS 3.3 APPEND command to tack succeeding small files onto the end of your main file. APPEND works just like the OPEN command, but it puts the position-in-file pointer at the first byte beyond the end of the file rather than at the beginning of the file. The DOS 3.3 APPEND command has some bugs, however (all solved in the March 1985 **Open-Apple**, page 24). In addition, using APPEND in an application like this would become quite slow as your main file got larger, because each time you execute APPEND it determines where the end of the file is by reading the file byte-by-byte until the end pops up.*

ProDOS, on the other hand, remembers the number of bytes in text files. The information is in the directory and can be seen by using the 80-column CATALOG command. The Basic.system APPEND command uses this information to go directly to the end of the file and is consequently much faster than DOS 3.3's version.

BLOADing DOS 3.3 text files

Val Golding asked in your February 1986 issue (page 2.6) for a way to BLOAD a DOS 3.3 text file, as under ProDOS. Here's one solution:

Before calling the routine below, ADR should equal the load address, and FILE$ should equal the name of the file loaded:

```
100  REM *** Routine to 'BLOAD' a text file ***
101  REM *
102  REM *  set ADR and FILE$ before entry

110  PRINT : PRINT CHR$(4);"OPEN ";FILE$;",L256"
120  DB=PEEK(46535) + PEEK(46536)*256 + 79
130  P1=PEEK(DB) : P2=PEEK(DB+1)
140  POKE DB+1,ADR/256
145  POKE DB,ADR - (PEEK(DB+1) * 256)

150  FOR REC=0 TO PEEK(46574)-2
160  PRINT CHR$(4);"READ";FILE$;",R";REC;",B0"
170  GET A$: PRINT
180  POKE DB+1, PEEK(DB+1) + 1
190  NEXT REC

200  PRINT  CHR$ (4);"CLOSE"
210  POKE DB,P1: POKE DB + 1,P2: REM restore pointer
220  RETURN
```

The routine works by fooling DOS into thinking its data buffer is at ADR, rather than above HIMEM as it usually is. Lines 150 to 190 load the file into memory, one sector at a time, while moving the apparent location of the DOS buffer higher for each sector. An important element of this trick is setting the record length to 256 (the length of a sector) and using GET to force DOS to read the next sector into the buffer.

Drawbacks? Sure, only sequential text files can be loaded since the GET command will cause an END OF FILE error if it finds empty bytes in a random access file. The routine loads whole sectors at once so any unused bytes at the end of the last sector will overwrite as many as 254 bytes in memory. DOS text files are stored and, with this routine, loaded with the high bit set, which may not be compatible with your routines. Still, it has its uses, and doesn't require any machine language.

Frank G. Andrews
Kalamazoo, Mich.

found group FORTH

Reference your mention of FORTH, (page 2.11), March 1986, ***Open-Apple***, fast it is! But the notation, Polish, never for me has been a problem! More natural, what could be? For years, using it I have been. The FORTH Interest Group, P.O. Box 8231, San Jose, Calif 95155, interested readers should write.

Anthony D. Alley
Eldorado, Texas

Applesoft compilers

It seems to me the reasons listed in your June issue (page 2.39) for not developing a ProDOS-based Applesoft compiler are really arguments against production of an *inadequate* compiler.

Speed is addictive, and I have yet to hear anyone complain that a given combination of hardware and software runs too fast. So the existence of other improvements that make programs speedier hardly renders compilers obsolete.

Since most of my own computing consists of various calculations I need for my physics research, I don't need to look far for program bottle-necks—floating-point arithmetic is inevitably what slows my programs down. I've already found out that Microsoft's DOS 3.3-based TASC compiler doesn't speed these calculations up at all. What's more, ZBasic does them more slowly, even with precision configured to fewer decimal digits than Applesoft. I'm unaware of really fast floating-point routines that one can insert into an Applesoft program. The floating-point boards from California Computer and Applied Informatics provide speed at the expense of precision and they're not cheap. So far, Applied Engineering's Transwarp board offers the easiest way to make my calculations go faster (ten minutes to do certain integrals instead of half an hour).

The one thing I envy IBM (and IBM clone) users is their option of simply plugging an 8087 chip into a socket to obtain dramatic improvements in floating-point performance. To be sure, an 8087 is not cheap as chips go, but it still costs less than accelerator boards and much less than floating-point boards. Still, my Apple does so much else well that I'm willing to walk away and let it churn out calculations in its own good time.

What the Apple II needs more desperately than a good compiler is an improved Basic. For all the wonders that clever Apple users have managed to bring us, from named subroutines with local variables to neat screen management commands, Applesoft is not up to snuff anymore. Reliable and fast floating-point arithmetic, multi-line functions, nice long variable names (with all characters significant, of course)—these and other features ought to be brought into standard Applesoft. The language that was great on 48K or smaller-capacity machines in 1980 is too confining now. Even Microsoft's GW BASIC for MS-DOS machines is better.

Harvey S. Picker
Hartford, CT

I agree, Applesoft is a limited language compared to almost everything else in use today. On the other hand, it has two strong points—it is readily available on almost every Apple II ever manufactured and it's easy enough to learn that almost everyone who is interested in Apple IIs and programming understands it.

This universality, unfortunately, makes it impossible for Apple to update the language. An updated Applesoft would simply not be Applesoft. The incompatibility problems that would inevitably arise would be worse than the limitations of today's Applesoft.

*I expect Applesoft to be our universal language as long as there are Apple IIs, but I also expect to see an explosion of more advanced languages for the Apple II during the next 18 months. Already we have **Kyan Pascal**, **ZBasic** (which you mentioned), and **Micol Basic** (which comes up in the following letters). More languages are on the drawing boards.*

Besides being universal, Applesoft continues to be an excellent language for beginning programmers. It is a good, productive environment for learning about computers in general. I don't buy the widely-promulgated opinion that Applesoft gives beginners "bad habits." What is the best way to learn the importance of good program structure—by being forced to use structure without explanation or by writing programs with poor structure and then being forced to enhance them? Letting students crash into Applesoft's limits is the best way to teach them the advantages of more advanced languages.

And, as I said, advanced languages for the Apple II are suddenly budding out all over. Meanwhile, here's a bud for you.

According to a reliable source, our friends at Beagle Bros have a ProDOS-based Applesoft compiler under development. The source, Uncle Louie, says it will compile any reasonable Applesoft program, although some ampersand commands will have to be changed. It works by simply adding a COMPILE command to your programming environment. You type COMPILE FILENAME, and your Applesoft program will be compiled and executed on the spot. You can also type COMPILE FILENAME, NEW.FILENAME and a compiled version of the program will be stored in a file with a special file type. Uncle Louie says the compiled file is usually shorter than the original Applesoft file.

Compiled files will be able to be distributed as commercial software. They will require the compiler's "run time" package, however, which will be licensable from the program's author, Alan Bird. Uncle Louie says he thinks the program will sell for $74.95 and will be available just before Christmas.

LIST protection

I've enclosed an ad from *Call-A.P.P.L.E* concerning a compiled Basic for the ProDOS environment called Micol Basic. The ad is a bit enthusiastic, but not too far from the truth.

Now, about your statement in the June issue concerning what half the world thinks about compilers and LIST protection...

I develop custom software for vertical markets like TV studios and audio-visual producers. Compilers, such as *Micol Basic*, are some of my most important tools. I compile because it reduces program development time (compared to using machine language) and the code runs faster that any interpreted Basic. Since most compilers offer chaining, I've never had to be too concerned with code expansion. I'd rather develop and debug modules anyway.

And managing dozens of utilities that all compete for the ampersand vector stopped being fun a long time ago. Besides, most of these utilities are copyrighted and give me no legal right to use them in software I develop for sale.

(By the way, I debug the Applesoft *before* compiling. There's no reason to have to compile over and over again.)

But the main reason I compile is for my own protection. This means including a front page in the program that states my copyright, the name and address of the licensee, and the terms of the license. If the program is copied, this information goes along with it. I've discovered that folks are very protective of software that can be traced back to them.

As a developer, I owe my clients workable object code that they can back-up for their own safety. I don't feel that I owe them the right to get into my source code and remove what little protection I have.

Frank Jaubert
Houston, Texas

Embedding a purchaser's name within a copy of a program is an unobtrusive copy-protection technique that is ideal in situations where you know who your customers are. I like it. If I were concerned about illegitimate copies, however, I would encrypt the purchaser's name so that it couldn't simply be changed to blanks with a disk zap program. Compilation isn't strictly necessary—an ampersand routine or even some obscure lines of Basic would protect the embedded code nearly as well.

Micol Basic

In the June issue of **Open-Apple** Robert Heldreth asked about ProDOS compilers. I am not aware of an Applesoft compiler for ProDOS but there is a program that I recently purchased that just about fills the bill. The program is *Micol Basic*.

Let me preface my impressions of the program with the caveat that I am not an expert. I bought my first Apple in 1980 (I was 45) and taught myself to program with *The Applesoft Tutorial*. With the help of magazines such as ***Nibble***, ***Softalk*** and others I have been able to write spectacularly unsophisticated programs for my own amusement and amazement. I gave up on assembly language. Here's what I think of *Micol Basic*.

I found the manual very well done but a few more examples would have been helpful. Especially IF/THEN BEGIN/ELSE BEGIN.

I like the added commands, such as FOR/UNTIL, PERFORM/UNTIL, REPEAT/UNTIL, WHILE/WEND, IF/THEN/ELSE, and the ability to GOSUB/ROUTINE NAME rather than GOSUB/LINE NUMBER. I think that the Micol disk commands, which don't use control-D, are superior to Applesoft.

Compiling your programs to /RAM is very fast. It took me awhile to figure out how to do it but with the help of PROSEL, a terrific program you've mentioned before (March 1986, page 2.10) from Glen Bredon ($40, 521 State Road, Princeton, NJ 08540), it's a piece of cake. I don't have a hard disk but Glen's program works great with the extended 80-column card on my IIe. Since I write programs by the old "trial and error" method, compiling to /RAM has saved me a lot of time. Compiling to disk for every one-character change you make to your program can put you over the edge very quickly.

The disk is not copy-protected. This is a big plus in my book.

Support for the program has been excellent. Call Micol Systems and you get the author.

The only weakness for me is the editor. Because of my "brute-force" method of programming, it would be nice if the editor contained a renumber facility. A *Micol Basic* renumber program is included in the Appendix of the manual but I find it too time-consuming to leave the editor, renumber, and then return to the editor.

I believe strongly in supporting programs that give you your money's worth. My experience shows that few programs live up to the claims of their authors or marketing people. *Micol Basic* is an excellent program at a very fair price and delivers as promised.

Jack Cowly
Temple City, Calif.

*As you mention, **Micol Basic** is not strictly an Applesoft compiler. It is an enhanced version of Basic with an Applesoft-like syntax. It is priced right at $49.95 plus $5 shipping and handling (Micol Systems, 9 Lynch Rd, Toronto, ONT, Canada M2J 2V6 416-495-6864). Our technical wizard Dennis Doms has taken an extensive look at **Micol Basic** and here's what he thinks:*

*Micol Systems claims in its advertising that "**Micol Basic** is capable of compiling your existing Applesoft programs...." As long as you take "capable" to mean something less than fully compatible, you will not be fooled by the ad. Most non-trivial Applesoft programs have to be modified somewhat before they can be compiled with Micol Basic. The modifications center on differences in disk I/O commands. There are also a few syntax differences and the compiler requires a few commands of its own. In addition, Applesoft files have to be converted to text files before compilation can begin.*

***Micol Basic** supports significant enhancements to Basic such as the structured programming commands WHILE-WEND, REPEAT-UNTIL, BEGIN-ENDIF, and so on. **Micol Basic** also supports "typing" of variables (you can define all variables whose names begin with the letters I-N to be integer variables, for example). A new variable type allowed by Micol is boolean. In addition, **Micol Basic** supports long variable names such as BLOODY_LONG_NAME (this feature can be disabled), and CHAINing of programs.*

Some of the syntax changes are cosmetic, probably to ease the job of the compiler program. One of the more prominent of these is that the arguments of VTAB and HTAB have to be in parentheses or you will get an error during compilation. Also, DATA must be declared prior to its reference by READ, immediately after any compiler options (so you wanted structure in your programs?...).

While browsing through the manual and reading about these features, I started getting enthused. The "loss" of Applesoft compatibility may be compensated by the extra features: speed and structure.

The development cycle is the usual compiler shuffle—EDIT the text file that holds the program source code, save the source file, COMPILE the source file to an object file, LINK the object file to the compiler's library to form an executable object file, load the run-time support package and EXECUTE the executable object file. In case of an error at any stage, go to the beginning and start over. Nonetheless, I found the Micol environment easy to work in.

To bring up Micol from a subdirectory you must call it by the complete pathname leading to it. For example, on my disk I had to use "-/PROG/MICOL.BASIC/MICOL.SYSTEM." Doing the normal ProDOS move of setting the prefix to "/PROG/MICOL.BASIC" and then issuing "-MICOL.SYSTEM" exiled me to the ProDOS quit prompt. This and a 32-character limit on filenames used within programs gives the impression that Micol's author, Steve Brunier, isn't yet sold on subdirectories.

*I transferred a copy of LITTLE BRICK OUT (a classic Applesoft game supplied on the old DOS 3.3 System Master) over to my work disk and saved it as a text file (if you don't know how to do this, the Micol manual explains how). LITTLE BRICK OUT was chosen as a guinea pig for conversion since it was mentioned in the **Micol Basic** manual and in a **Softalk** (Sept. 1981, pages 95-102) article evaluating Applesoft compilers. It is also a moderate-sized program; after converting it to ProDOS it used 15 blocks. Converting it to text gave a file size of 22 blocks. Editing it to facilitate compilation added one more block. I did the minimum amount of editing needed to allow the file to compile and run. This primarily entailed changing VTAB and HTAB statements, re-defining logical IF statements of the type "IF var THEN" to comparisons of the type "IF var<>0 THEN", and keeping LITTLE BRICK OUT's POKE statements from tromping on Micol's use of 768-815 ($300-$32F).*

*The Micol editor is a limited but usable line-oriented text editor. I used the editor's F)ind and R)eplace commands to find most syntax problems. Neither command includes a wildcard option so trying to find and replace all occurrences of VTAB <number> for all possibilities of <number> got a bit tedious. It's easier to do conversions from Applesoft by using **Apple Writer** to do the searching and replacing. (In fact, a good WPL program could turn*

__Apple Writer__ into a stand-alone preprocessor.) An immediately needed change in is the ability to renumber lines from within the editor (yes, __Micol Basic__ still requires line numbers for all lines).

When I got tired, I let the compiler try to find the remaining offenses. During four compiles I noticed that compile errors for certain lines were not always reproducible. One example: during the 3rd compilation I got an error in line 970 (for an improper "IF" syntax) that was not caught in the second compilation. Line 2947 had a missing ";" between strings in a print statement. Line 3010 gave an error over something about a "print statement" (there is no PRINT statement in line 3010 of LITTLE BRICK OUT). I corrected lines 970 and 2947 and fed it to the compiler a fourth time, hoping for a recurrence of the message given for line 3010. This time it compiled with no errors to an output code file of 20 blocks.

In addition to that 20 blocks, we have to have a 21 block run-time program called LIBRARY installed. This replaces the ProDOS command interpreter, Basic.system. Replacing Basic.system with the library is good in the sense that it saves memory space, but it's bad in the sense that you can't enter ProDOS commands on the keyboard when you exit a __Micol Basic__ program. And if you enter a ProDOS command by mistake at that point __Micol Basic__ sometimes crashes.

For what it's worth, __Softalk__ reported that Microsoft's DOS 3.3 TASC compiler produced a 39 sector file (about 20 blocks) when compiling LITTLE BRICK OUT, and required a 17 sector (9 blocks) run-time package (in addition to all of DOS 3.3).

The edit/link/compile/execute procedures went very quickly. The program linked fine and ran fast. Once execution has finished, you use the "&" command to return to __Micol Basic__.

Thus encouraged, I decided to run a benchmark to check the speed. I chose Jim Gilbreath's "sieve" benchmark from __Byte__ ("Eratosthenes Revisited", Jan. 1983, pp. 283-326). The IBM Basic version seemed suited to __Micol Basic's__ extra constructs. I typed it in, adding a line for the program name and changing "10 DEFINT A-Z" to Micol's "10 INT (A-Z)." I called the compiler. It told me I was out of simple array space in line 10.

It hadn't registered with me that the assignment of variable space was not fully dynamic; one of the compiler options allows (nay, requires for this program) the programmer to assign pointers that determine how much memory space is to be reserved for simple variables, simple strings, variable arrays, and string arrays. I adjusted these to give simple arrays the lion's share of memory. The program compiled, and executed in 476 seconds. For comparison, I timed the same program under standard Applesoft at 3764 seconds. That, folks, is a good improvement, even considering the fact that benchmarks sometimes don't tell the whole story.

__Micol Basic__ also has a QUICK compiler option, which produces code somewhat optimized for speed at the expense of some memory efficiency. The sieve program, compiled using this option, ran in 444 seconds and was 429 bytes in size (before linking) compared with 476 seconds and 393 bytes for the standard mode.

I did a simple FOR-NEXT loop from 1 to 10000 in __Micol Basic__ and Applesoft. With a floating-point counter, Micol took 10.5 seconds (versus Applesoft's 19.9); with an integer counter it took only 4.1 seconds (Applesoft does not allow integer counters in FOR-NEXT loops). The Micol manual suggests using integer variables when possible for maximum speed.

Finally, I did a test with the Savage benchmark from the __Byte Special IBM Issue__, Fall 1985 (page 70). This benchmark is supposed to test the efficiency of floating-point math routines. Under standard Applesoft the test took 472 seconds; Micol took 463 seconds. As you can see, Micol does not greatly speed up floating-point operations.

The Applesoft/__Micol Basic__ syntax incompatibilities that exist and the necessity of switching system interpreters to go from __Micol Basic__ to Applesoft makes Micol less than ideal for what I perceive most people want an Applesoft compiler for—to quickly compile an interactively debugged Applesoft program.

However, looked upon as a separate language from Applesoft, Micol becomes a lot more attractive, especially at the price. Even the licensing fees for using Micol's run-time package in commercial software are low—free with screen credit to Micol. Micol's nearest competition in Appledom at present is __Kyan Pascal__, which is more sophisticated but also has been around longer. Micol is still Basic, line numbers and all, which may be a plus in finding an audience. For someone who wants to stay with a dialect of Basic, who wants speed, and who wants the ability to write structured programs, Micol is definitely a noteworthy and inexpensive solution.

ProBasic, Logo

I have not yet seen in your pages (nor hardly anywhere else) mention of the fine programs coming out of The Software Touch, by the talented authors Mark Simonsen and Alan Bird. In particular, Alan's *ProBasic* (distributed as a "freebie" on the back of *Program Writer*) bids fair to totally transform the experience of programming in Applesoft.

ProBasic allows you to add new commands and functions called modules to your programs. Each module has its own set of local variables, so variable conflicts are easier to avoid. Parameters can be passed to the modules by value or by reference (even by reference to another module). The modules can be written in either Applesoft or assembly language —thus adding a natural, consistent interface for machine language routines (look Ma, no &).

That's power, and it's pretty! Best yet, there is nothing to unlearn—nearly every bit of the special Apple II lore I have accumulated over the years is still relevant. Programming in ProBasic is the next best thing to programming in APL (another story, for some other time), or at least it promises to be if Alan can get the bugs out.

I have one question to ask, concerning sources of information on *Apple Logo II*, which you mentioned once or twice in **Open-Apple**. Two years ago I wrote a review of this product for (alas) the now-defunct ***Microcomputing***, and in the process acquired a review copy. I have never since, however, seen a whisper of published evidence that anyone is developing applications in *Logo II*. I dig out my copy now and then and play with it a bit, and admire it all over again. The main problem I have is that my computer is an Apple IIe and my printer an Epson MX-80 and I have not been able to find any way to generate hard copy directly and conveniently (the name of the game!). I would greatly appreciate your pointing me to a printer driver I could patch in, or even a few clues as to how I could approach developing such myself. I am not a technical whiz, but could it be that this is another case of a project that would not be impossibly complicated if just a little open information were available?

R. W. W. Taylor
Rochester, N.Y.

It's interesting that you mention __ProBasic__ and __Logo__ in the same letter. The way __ProBasic__ allows you to write modules and call them by name with parameters is very Logo-like.

To tell you the truth, I hadn't noticed __ProBasic__ (hidden away as it is on the back of __Program Writer__) till I got your letter and I still haven't had a chance to use it, but the documentation is absolutely startling.

Not only does __ProBasic__ provide modules, it has a command called VIRTUAL that allows arrays (both numeric and string) to reside in disk files. This means the size of an array is limited only by the amount of room on your "disk." You wouldn't want to use a floppy for a VIRTUAL array, but this is one good way to use large /RAM disks from Basic. A hard disk is also a possibility for really large arrays—and you don't have to mess around saving the data in the array in a separate file.

Bird and Simonsen started The Software Touch about a year ago. They are well known as the authors of such Beagle Bros classics as __Beagle Graphics__, __Extra K__, __Double-Take__, and __D Code__.

They currently have four major products. __Font-Works__ ($49.95) reads and prints standard text and AppleWorks files in a variety of fonts, type styles, and sizes. It can print spreadsheets sideways. It has an AppleWorks-like interface and comes in both ProDOS and DOS 3.3 versions. It runs on any Apple II.

__Graphics Pro__ ($29.95) is a single high-resolution graphics program like __MousePaint__, only better. It allows you to work with the entire screen, supports graphic tablets and joysticks as well as mice, and, unlike __MousePaint__, can show you a disk catalog.

__AutoWorks__ ($39.95) adds macros to AppleWorks. The macros can be updated from within AppleWorks, can use conditionals, can contain lists (where each time you use the macro it takes the next item from the list), and support date and time stamping. The program includes a built-in mail merge capability, adds mouse control for cursor positioning and menu selection, and has a file organizer that reads the filenames from all the directories and subdirectories on a ProDOS disk into an AppleWorks data base file. Alan is working on, but has not yet accomplished, __Pinpoint__ compatibility.

__Program Writer__ ($49.95), besides including __Pro-Basic__ for free on the back, is an AppleWorks-like full-screen editor for Applesoft. Using __Program Writer__ is a lot like writing a program using the AppleWorks word processor, except that it has additional features such as the ability to renumber lines, the ability to list all the variables in the program you are editing, and the ability to instantly disappear, leaving your program in memory ready to run. After running your program, entering &&& returns you to the editor with your program intact.

In addition to these four main products, The Software Touch also has two disks of additional modules for __ProBasic__ at $20 each. Their stuff is available at about 200 dealers around the country, or by mail order (9842 Hibert St., #192, San Diego, Calif. 92131 619-549-3091).

I asked Alan what kind of bugs you were talking about in __ProBasic__. He said an early version had some bugs in the VIRTUAL command and in the code that saves modules on disk. These have been fixed—if you send him your original disk and $5 with a note that you want __ProBasic__ updated, he will take care of *it for you.*

*I'm not sure what your printer problem is with the Epson/**Apple Logo II** combination. The **Logo II** command for turning on a printer is OPEN 1 SETWRITE 1. If your printer isn't attached to slot one use the correct slot number instead. Any PRINT, TYPE, or SHOW statement will now go to your printer. Return to the screen with CLOSE 1 SETWRITE [ ].*

*If what you want to do is print a listing of your program, however, OPEN 1 SETWRITE 1 doesn't work. Instead, use DRIBBLE 1 POALL NODRIBBLE. The **Apple Logo II** documentation gets a negative 5 stars for its explanation of this one (I had to call tech support at Logo Computer Systems, developers of **Apple Logo II**, in Quebec—514-631-7081—to find the solution, which is hidden on page 210 of the reference manual under the discussion of DRIBBLE).*

*Some Logo users have had problems with line length when listing programs. These problems can sometimes be fixed by changing the interface card setting for line length with a control-I command. Use a Logo command line such as (TYPE CHAR 9 "XX CHAR 13), with XX equal to the command you want to send. See your interface card manual for the command you need. For the Apple Super Serial Card try "C." Unfortunately, you can't send commands to some interface cards, such as the Grappler, because **Logo II** prints CHAR 9 (control-I) with the high bit clear and the Grappler expects the high bit to be set. The November 1985 issue of **Open-Apple** has more information on this problem.*

*To discover why no one is developing applications in Logo, load and run the sample programs that come with **Apple Logo II**. 6502 versions of the language simply can't execute fast enough for typical computer applications. Note that the disk is copy protected and that Apple has made no provisions for licensing a **Logo II** "run-time" package. This means anyone who purchased a **Logo II** application would also need to buy **Logo II** itself. The primary use of Logo is in education, where most people consider it not so much a computer language as a language or model for learning.*

*If you're interested in further information on Logo, check out **The National Logo Exchange: The Logo Newsletter for Teachers** ($25 for nine monthly issues, Sept through May, P.O. Box 5341, Charlottesville, Va. 22905). It has been published since September 1982 and is the most comprehensive Logo publication I know of. Its long suit is educational theory but it also includes solutions to programming and technical problems.*

The ampersand solution

Anyone who does any programming at all shouldn't miss Roger Wagner Publishing's offer of a free copy of the *The Trial Size Toolbox* made in the April **Open-Apple**, page 2.22.

I had a very interesting talk with Roger the other day, and I thought many of **Open-Apple**'s readers would be interested. I had noticed that not all of the original *Routine Machine* series was upgraded into *Toolbox* packages, so I called to find out why.

Apparently the return rate on registration cards from the old *Routine Machine* programs was rather low. Many original owners may not realize it, but Roger Wagner Publishing does have an upgrade offer for original purchasers. Even if you haven't mailed in the registration card, you can mail in the title page from the manual as proof of purchase along with $20 (plus $3 shipping) and benefit from the upgrade. The *Toolbox* series is not protected, as was the original *Routine Machine*, and several new routines have been added and others upgraded.

The *Toolbox* series is designed for programmers! Roger says that large ads in computer magazines do not provide the same return for this series as they do for application programs (i.e. *MouseWrite*). The result of all this is that a limited number of people have purchased the *Toolbox* packages, either new or through updates, and Roger has no plans to release further disks in the *Toolbox* series, even though they have over 150 additional routines that could be released.

So fellow programmers, order those disks, and tell Roger that there is an audience out there that is interested in more routines. Maybe we can convince him to release another disk. I for one want to be first on the mailing list, as I have been using many of these routines for some time now, and wonder how I could program without them. The routines give a professional appearance to your programs while simplifying the coding.

Open-Apple is directed at the same audience as the Toolbox series. So if you are reading this and you don't presently use the Toolbox, you would probably be interested in these programs. There are routines for all those items you would like to have in your program: IF-THEN-ELSE, PRINT USING, CATALOG READ ROUTINES, INPUT ROUTINES, OUTPUT ROUTINES, SCREEN FORMATTING, SORTING ROUTINES and more.

Don Druce
Longueuil, Quebec

*Roger's offer of a free **Trial Size Toolbox** to any **Open-Apple** reader who writes in and asks for one is still open. (Roger Wagner Publishing, P.O. Box 582, Santee, Calif. 92071) The four toolbox packages currently available are **The Wizard's Toolbox**, **The Database Toolbox**, **The Video Toolbox**, and **The Chart 'n Graph Toolbox**. Each sells for $39.95.*

*The routines can be used in commercial software without a licensing fee. You do have to write in for a formal license agreement, however, and include a credit line in your finished software. The **Toolbox** packages are shipped on DOS 3.3 disks, but all readily CONVERT to ProDOS with the exception of the **Chart'n Graph Toolbox**, which is available by request in a ProDOS version.*

*Routines from the various **Toolbox** packages are all compatible with each other. They share the ampersand hook. The beauty of the **Toolbox** series is the way the ampersand hook and memory requirements are automatically handled by the package. The necessary machine language code is attached to the end of Applesoft programs where it stays permanently. No BLOADS, buffer moving, or GETBUFFER calls are required.*

Another place for code

In the July issue (pages 2.43-44) you "released the power to everyone" by showing how to embed machine language in a REM statement as an alternative to using memory page three. You also mentioned that method was ONE way to do it. I think it's a little too restrictive—the machine code cannot contain a zero, it cannot exceed 239 bytes, and it LISTs as garbage on the screen. I have always hidden my machine language code at the end of a program, between the last line and the variables.

I can't take credit for inventing this method, but I will do my part in "releasing the power" and share it.

The only restriction that I know of with this method is that the machine code must be relocatable (no reference to locations inside itself). It will work with either ProDOS or DOS 3.3.

```
1. Clear the machine:
      NEW (or FP under DOS 3.3)

2. LOAD your BASIC program:
      LOAD MY.PROG (you can skip this step if
                   your program isn't written yet)

3. Get the current program end:
      PE = PEEK(176)*256 + PEEK(175) : PRINT PE

4. Load the machine language code onto the end:
      BLOAD ML.CODE,AXXXX
              (use the value from step 3 for XXXX)

5. Calculate the new program end:
      PE = PE + length of ML.CODE

6. Change the pointer to the new program end:
      POKE 176, PE/256 :
      POKE 175, PE - PEEK(176)*256

7. Save it:
      SAVE MY.NEW.PROG
```

You may add, delete, or edit your Applesoft lines as you wish. You should be able to use your favorite line editor as usual. You can use EXEC to add program lines or even to add whole programs to a machine code skeleton file.

As you edit the Applesoft program, the machine language segment is moved up and down in memory automatically when the program gets longer or shorter. To find its address during program execution, add this line when you set up your variables:

```
MLADR = PEEK(176)*256 + PEEK(175) -
          length of ML.CODE
```

Then, when you need to CALL the routine, simply CALL MLADR.

Steve Stephenson
Checkmate Technology, Inc.
Tempe, Ariz.

For those of you who have been following my series on tricks with Applesoft in the April, June, and July issues, here's a little more detail on how this method works.

Look at one of the memory maps from those issues—on April's page 2.19 or June's page 2.35. In this technique, the machine language code is added to the end of the "program image"—the second area from the top in the memory maps.

In between the Applesoft program and the machine language code there are three zeros. This method works because some parts of Applesoft consider the three zeros to mark the end of the program while other parts use the PRGEND pointer at 175-176 ($AF-B0) to mark the end of the program.

The genie that executes programs uses the three zeros. Thus, that genie never tries to execute the machine code as if it were Applesoft tokens. The genie that corrects the "next-line pointers" also uses the three zeros. Thus, that genie doesn't go searching through the machine code looking for the zero bytes that mark the end of Applesoft lines. If she did, she would try to "fix" the next-line pointers and, as a consequence, trash the machine code.

On the other hand, the genie that moves Applesoft programs up and down in memory as lines are added or deleted uses PRGEND. Because of this, the machine code segment essentially becomes "stuck" to the end of the Applesoft program image. The SAVE and LOAD commands also use PRGEND, so the machine code segment automatically follows the program in and out of disk files.

The only difficulty with this technique is that if you have more than one piece of machine code, you have *to keep track of the exact length of each and do some calculations to figure out where in memory they end up. Roger Wagner's **Toolbox** series automates this difficulty and makes it easier for routines written by different people to share this memory resource.*

The Integer version

Since I'm probably one of the few people around that even remembers there was such a thing as Integer Basic, I thought I would add to your reply to the letter from Len Lipshultz (July, page 2.44) about the mysterious appendix at the end of *APPLEVISION*.

An Integer BASIC program is stored downward in memory from the highest available location, less one, as defined by HIMEM: (76-77, $4C-4D), while the variable table builds upward from $0800. In Integer, when the HIMEM: command is used, a memory move is executed and the program is relocated with its last byte at the new location of HIMEM:-1.

To attach a machine language program, just calculate the amount of space required for the code to be added and lower HIMEM: appropriately using the HIMEM: command. Then type in or BLOAD the binary data. Remember where it starts as this becomes the permanent starting address for the routine. Finally restore the HIMEM: pointer to its original setting—this time using POKE rather than the HIMEM: command—and save the program. When you list the modified program you will, of course, see a bunch of garbage at the end.

Val J. Golding
Tarzana, Calif.

Open-Apple

is written, edited, published, and

© Copyright 1986 by
Tom Weishaar

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

Most rights reserved. All programs published in ***Open-Apple*** are public domain and may be copied and distributed without charge (most are available in the MAUG library on CompuServe). Apple user groups and significant others may obtain permission to reprint articles from time to time by specific written request. Requests and other editorial material, including letters to Uncle DOS, should be sent to:

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

Problems with splits

Your article on splitting Applesoft programs in the April issue was great (pages 2.17-2.21). It's hard to imagine that there could be any more to say on the subject, but here are some solutions to problems I've encountered with split programs.

In *All About Applesoft* (page 118), David Lingwood points out that Applesoft does a limited check of its current position in a program when a GOTO or GOSUB is executed. If the high byte of the destination line number is greater than that of the current line number, Applesoft begins searching for the new line from the present line, rather than going all the way back to the beginning of the program.

In your example, line 9999 changes its own next-line pointer and then says GOTO 10000. Change this to GOTO 10240 or more and the program will crash with an UNDEF'D STATEMENT error. You can be sure that a split will work with any combination of line numbers (as well as get a bit faster operation) by ending the first segment with a dummy line. The only function of this dummy line is to hold a modified next-line pointer that's aimed at the second program segment. Try these modifications to your example at the top of page 2.21:

```
9980 ADR = 16384 : REM adr of overlay
9981 HI = INT((ADR+1) / 256) :
     LO = (ADR+1) - (HI*256)
9982 HERE = PEEK(121) + PEEK(122)*256 :
     NXT= PEEK(HERE+1) + PEEK(HERE+2)*256 :
     POKE NXT,LO : POKE NXT+1,HI :
     GOTO any line number in second segment
9999 REM dummy line
```

Another problem is that the DATA pointer at bytes 125-126 ($7D-7E) won't jump the gap between the two parts of a split program. You have to help it across—otherwise trying to read a DATA statement that's in an overlay will return an OUT OF DATA error. You can reset the DATA pointer when you load an overlay by changing line 9980 (your line 9997) to:

```
9980 ADR = 16384 : POKE 126, ADR/256 :
       POKE 125, ADR - PEEK(126)*256
```

This method works fine when both the READ and DATA statements are in the second segment of a program. If, however, you were simply splitting a single big program to straddle the graphics page, the split might occur such that a READ in one segment refers to DATA in the other segment. In that case, a better trick would be to adjust the DATA pointer in an error-handling routine:

```
9000 IF PEEK(222) = 42 THEN
       DP = PEEK(125) + PEEK(126)*256 :
       IF DP < ADR THEN POKE 126, ADR/256 :
       POKE 125, ADR-PEEK(126)*256 : RESUME
```

A final note—there is a very handy Monitor routine at -327 ($FEB9). It allows you to call assembly language routines from Applesoft with the 6502's registers loaded with whatever values you wish. POKE the value you want in A into byte 69 ($45), X into 70, Y into 71, and P into 72. Put the address of the routine you want to execute into bytes 58 and 59 ($3A-3B). Then just CALL -327. This entry point is part of the Monitor's GO command.

Paul Nix
Summit, NJ

A new format

How can I format a ProDOS disk from within an assembly language program?

Robert C. Moore
Laurel, Md.

There are two ways to go. Apple will license you its ProDOS FORMATTER routine for $50 a year per application—see our July issue, pages 2.41-42 for more information.

*Much more interesting is a public domain program called **Hyper.FORMAT** by Jerry Hewett. It comes with source code so you can modify it to your needs or include it in your own programs.*

***Hyper.FORMAT** is a product of Living Legends Software, which is a loose association of "user supported" Apple programmers who have pooled their resources and enthusiasm to establish market recognition. "User supported" software, also known as "freeware," can be copied and distributed by anyone without charge. Anyone who decides to keep and use a freeware package, however, is expected to pay the program's author. Bill Basham, author of **Diversi-DOS**, has used this method of distribution for years.*

***Hyper.FORMAT**, however, isn't freeware, but is one of several public domain programs the Living Legends authors have put together on a disk they call **Misk Disk #1** ($10 from Living Legends Software, 1915 Froude St., San Diego, Calif. 92107 619-222-3722). Two other programs on this disk, THE.EXECUTIVE and UNICOPY, were mentioned in a letter to **Open-Apple** in February (page 2.7). Living Legends Software's programs can be downloaded from the Apple II libraries on CompuServe, Delphi, and BIX, as well as many local bulletin boards. Or you can write them and ask for their catalog of interesting, reasonably-priced commercial programs.*

Basis 108 & Amiga

You have mentioned both the Franklin and the Lazer 128 in recent issues—there is another legal Apple-compatible on the market, although it is relatively hard to find in the U.S., the West German Basis 108.

Although Basis has no formal distributor in the U.S., our company acts as an importer and dealer to the many Basis user groups scattered about. We also act as a source and clearing house of both Apple and CP/M software and hardware that can use the full capabilities of the Basis.

You've also mentioned the Amiga, and I certainly won't argue with your assessment of Commodore's business health. As an old-time hardware engineer, however, I must say that the Amiga first struck me as the machine Woz would have designed if he had done the first Apple from scratch in 1985. It's certainly the first machine since the original II that shows the sort of engineering elegance, cleverness, and versatility that made our beloved Apples (and my Basis—grin) so near and dear to our collective hearts. If Apple had managed to get rid of Jobs earlier, maybe they, instead of Commodore, would have bought the Amiga from Intermedics. The thought of the Amiga with Apple's name, further developed by Woz and some of the others who were chased away in the yuppification of Apple brings tears to my eyes.

Bob Stout
Symbionic Systems, Inc.
P.O. Box 428
Alief, Texas 77411
713-465-9090