Open-Apple™

Releasing the power to everyone.

May 1985
Vol. 1, No. 4

My Two Bits

by
Tom Weishaar

InfoWorld's John Dvorak did a two-part series last fall on the characteristics of successful personal computers (Oct. 15, 1984, page 88; Oct. 22, 1984, page 80). After examining a number of features of successful machines and showing that all of them were available on the failures as well, Dvorak argued that the "real common denominator for success" is "heavy hobbyist enthusiasm and subsequent third-party software support."

Thus, Dvorak said, the successful personal computer, "has to have an easy-to-use programming language." He continued:

There is no successful machine (and there never will be in our lifetimes) that cannot be programmed in Basic, the people's language.

*Though Basic is **not** the world's greatest language, I submit that its ease of use makes it the perfect helpmate for the "guy with the idea." This means the nonprogrammer who has an idea for a program.*

*He writes the Basic program and shows (maybe sells) a compiled version—that is essentially the prototype. A sharpie can come in later and recode it. **This is the key** to third-party support—support by the masses.*

*Although professional programmers and **computer snobs** won't admit it, there are more good ideas among the 230 million U.S. laypersons than among all the professional programmers in the world. If the masses of enthusiastic laypersons (a subset of the general public) aren't encouraged to support a computer—it's dead.*

The Apple II, of course, has always had a wealth of third-party support. Dvorak calls it and the IBM-PC the "all-time supersuccessful machines." And he's right.

But nearly every personal computer ever built has been able to run Basic. I agree that any successful personal computer will have to include an easy-to-use programming language, and Basic is also my choice, but Basic hardly divides the successful computers from the losers. There is, in fact, another element that separates the machines that attract technically-inclined laypeople from the machines that can't find a market. It's not Basic and it's not any specific hardware feature, for many of the dead machines can match or better the successful ones in those categories.

This mysterious factor is the willingness and ability of the manufacturer to expose the inner flesh of a machine to light of day. Assuming a new machine has a reasonable amount of features and a reasonable price, its success with "hobbyists" (and consequently with the general public) will depend on the quality of the *information* and *development tools* that are available to *mere mortals.*

In the beginning, both Apple and IBM explained in great, readable detail how their machines worked. At one time Apple included this information in a reference manual that came in the box with its computer. The Apple II, of course, also came with built-in Monitor software (examined at length in the February and March ***Open-Apples***), which helped thousands of people to better understand how the machine worked.

Measure the latest, greatest products against this standard—the availability, *to the general public*, of the information and the tools needed to build hardware and software enhancements and to connect them to the operating system—and you'll be able to tell whether a machine will be a success or not.

If Apple had released a decent development language and decent documentation for the Macintosh the day they announced that computer, its sales goals would have been shattered. Even the Apple IIe and IIc would be in stronger positions today if the reference manuals for those machines had been more readily available the last two years.

While Addison-Wesley's deal to distribute Apple's manuals, reported here last month, should vastly improve the availability of information about the Apple II to the general public, the machine has also been hurt because newcomers with technical questions can't get support from Apple. Apple's current policy is that members of the general public with technical questions should go to their dealers.

What is the most elementary kind of technical support a dealer can provide? Stocking the reference manuals would be one of the easiest and most helpful things a dealer could do. But many dealers don't do even that. They say Apple makes them buy the books five at a time and this means an investment of a little more than $200 to keep the IIe and IIc reference manuals in stock, so they don't do it. Many even refuse to accept *special orders* for the manuals. Where does dealer-based technical support go from here? (For much more on this issue, see the Letters in the April 1985 *Byte*, page 23 and following.)

The dealers and Apple can argue all they want that the general public doesn't want detailed technical information. It is *obvious* that most computer users could care less whether the microprocessor inside their machines is a 6502 or a DC-10. But that still leaves *thousands* of people, none of whom are certifiable "software developers," who want technical information. The rapid success of ***Open-Apple***, the hundreds of user groups around the country, and the best-selling status of books like *Beneath Apple DOS* amply demonstrate that there is a significant market for detailed technical information about the Apple II.

I suspect the real reason dealers and Apple's marketing wizards don't recognize the size of this market is that they don't personally find technical information useful. Let's face it, most sales engineers (there are notable exceptions) are novice computer users. The result is situations such as the one that occurred with the *ProDOS Technical Reference Manual.* Apple's documentation department had it written, published, and ready to ship a full month before ProDOS was released to the public in January 1984. But by February the entire stock of books had been sold out. Apple's "System Software Product Marketing" department apologized in a letter to one of our readers for the difficulty he encountered in obtaining a manual and noted, "we underestimated the demand." Indeed, they did.

But the crux of this issue isn't that there is a *market* for techncial information, it is that *the long-term success of a computer is directly related to the amount and quality of technical information available to the general public.* It really doesn't matter whether those who desire technical information make up a significant portion of the market for a computer, the point is that *if a computer company doesn't get* **and keep** *the technically inclined layperson interested in its machine by providing for his or her needs, no one else will be interested in it either.*

The importance of the technically inclined layperson is twofold. First, these are the people the general public turns to for advice about personal computers. The word-of-mouth advertising these people can put behind a product is priceless. Secondly, these are the people who end up writing the software, both public domain and commercial, and designing the hardware that makes a personal computer worth having. (Consider, for example, the impact on Apple if Dan Bricklin and Paul Lutus, the two mere mortals who came out of nowhere to develop *VisiCalc* and *Apple Writer*, had written programs for Atari instead.)

What are the limits of the Apple II's success? In an interview in *InfoWorld* (April 15, 1985, page 34), Del Yocam, head of Apple's Apple II Division, said:

I have yet to find anyone willing to put an X where he believes the Apple II is in its product life cycle. We really believe the Apple II can live forever. But we also realize it is incumbent upon us to make that happen. We are putting $30 million to $40 million into research and development for the Apple II product family to make sure that we incorporate those things that are exciting and necessary to keep the machine's viability intact.

For eight years now, Apple has been able to enhance the original Apple II design and keep the machine endowed with a reasonable amount of features at a reasonable price. For eight years now, Apple has been able to keep the technically inclined layperson interested in its machine. Like Yocam, I really believe Apple could continue this forever (at least in computer-time), if it does things right.

Apple broke a considerable limitation to continued interest, that of confined external storage space, when it released ProDOS. Third-party developers have broken a second major limitation, that of confined RAM, with advanced memory cards. We'll take an extensive look at one of these and its implications in this issue of ***Open-Apple***. The third major limitation of the Apple II is its microprocessor, the 6502. Typical of people's feelings is this comment from one of our readers, "We ask a lot of our little 6502-based Apples. Is it my imagination, or do I really hear these little 8-bit machines panting as they execute integrated software packages such as AppleWorks?"

Apple appears to have enough *money* to break through the hardware limitations of the Apple II and appears, based on Yocam's comments, 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.

RAMworks transforms AppleWorks

Apple and its dealers have become very fond of selling the IIe as a "system" that includes disk drives, a monitor, and other peripherals. While the IIe itself is still one of the best computer buys anywhere, improved (and less expensive) versions of all that other stuff are widely available from third-party vendors. Notice that First Class Peripheral's 10 megabyte hard drive (reviewed here in February, page 10) is $30 *cheaper* than Apple's DuoDisk floppy and you'll quickly get the picture.

Right now Apple's competition is particularly fierce in the area of extended memory 80-column cards. Apple sells two versions of its own extended memory card. Both include 64K of RAM (bringing the IIes they're installed on up to 128K). The difference between the two cards is that one has the necessary connections for attaching an RGB color monitor to the computer (more about RGB later in this letter); the other doesn't. Since the price difference between the two cards is a mere $4 ($295 vs $299), it appears at first glance that the "AppleColor Card" is a steal not to be missed.

For just $179, however, you can buy a card from Applied Engineering (P.O. Box 798, Carrollton, Texas 75006; support and credit card orders: 214-492-2027, 8 am to 11 pm central time, seven days a week) that does what Apple's standard card does, but has a few extra capabilities. First of all, an RGB option can be added to the card later, when and if you need it, for $129 ($308 total; $9 more than the AppleColor Card).

More importantly, however, the amount of memory on Applied Engineering's card, which is called RAMworks, can be increased at any time (in 64K and 256K increments) up to 1 megabyte (1,024K). Apple's card, on the other hand, can't expand.

By expanding the amount of memory in your Apple IIe you can turn it from a meek little machine built around 8-year-old technology into an incredible powerhouse that amazes everyone. Old-timers may not believe this reincarnation is possible, however, because of a major problem extended memory cards for the Apple II have had in the past—there was precious little software around that actually used them.

People who bought these cards expecting Applesoft to allow larger programs were always disappointed; Applesoft was designed for a maximum of 48K of RAM and isn't ever going to be able to use more than that unless someone rewrites it. In fact, no high-level language for the Apple II that I know of has built-in support for more than 128K of memory.

This leaves essentially four ways to take advantage of an extended memory card.

First, you can write your own assembly language programs that use it. This is pretty rare, but it happens.

Less rare, but not by much, are commercial programs that look for extended memory and use it when it's available. A good example of existing software that does this is *MagiCalc* (sold by user groups under the names *IACcalc* and *The Spreadsheet*). The reason this kind of software is rare is that extended memory cards from different manufacturers are usually incompatible with each other. Up till now, no single manufacturer has sold enough cards to set a standard.

Third, you can use an extended memory card as a RAMdisk. This makes the card act like a very fast disk drive. Such drives have their limits, however—data stored on a RAMdisk can't survive a power failure. RAMdisk users have to copy all the files they're working on to the RAMdisk after turning the computer on, and, if the files are changed, save them back to floppy before turning the computer off. An additional problem is that most commercial software, especially the copy protected kind, provides no way to connect or use a RAMdisk, even when lots of RAM is available. On the other hand, a RAMdisk *can* be used to enhance the performance of high-level languages. One way to create a large Basic program is to break it into pieces, store the pieces on a RAMdisk, and use Basic.system's CHAIN command to switch among the pieces when necessary. If switching is kept to a minimum, the program will execute only slightly slower than normal, because RAMdisks are so fast. This technique can also be used with DOS 3.3, but the procedure for CHAINing under DOS 3.3 is more complicated.

The fourth way to use extended memory is to modify existing commercial software so that it recognizes and uses all of the memory available. Applied Engineering's RAMworks comes with a disk of modifications for *AppleWorks*. It is this particular combination of *AppleWorks*, RAMworks, and Applied Engineering's AppleWorks patches that suddenly make memory expansion on the Apple II a wonder to behold.

To understand the power of this combination, you have to know a little about *AppleWorks*. *AppleWorks* provides its users with a "desktop" onto which files can be placed. These files can be spreadsheet files, word processor files, or data base files. Up to 12 files can be placed on the desktop at once. By using a built-in "clipboard", data can be moved between files of the same type. Data can also be moved from any kind of file into the word processor. But more importantly, when two files are on the desktop, a user can switch between them without having to waste time saving or loading either. If you've ever been working on a document with a word processor and needed to refer to numbers in a spreadsheet, you know how time consuming switching data and programs can be.

But with *AppleWorks* and RAMworks, the switch can be instantaneous. Here's a real example. Over the past few years, I've developed spreadsheet templates for the various income tax forms I have to file. This year I was able to load all these templates onto the *AppleWorks* desktop at once and switch between them instantly. This gave me the ability to enter figures and fix mistakes in all relevant templates at the same time, rather than trying to remember the changes I'd made on little scraps of paper. This saved me *hours* of bewilderment.

Reviewers of *AppleWorks* have tended to concentrate on what it can't do (that stand-alone packages can) rather than on the synergistic effects of having three powerful programs available at the same time. I still use the more-powerful *Apple Writer* for most word processing chores—but I write for a living. Each of *AppleWorks'* modules has plenty of capability for most of us and more than enough for the rest of us.

With a 64K extended memory card in place, *AppleWorks* has a 55K desktop. With a 128K card, the desktop increases to 101K.

The real power increase comes with a 256K RAMworks, however. At this point you not only have a 200K desktop, but the entire *AppleWorks* program can be loaded onto the card as well. Start-up time increases under this option, but *response time once the program is running is immediate.*

Applied Engineering sells its card with 256K of RAM already installed for $399. The top-of-the-line card, with 1 megabyte of RAM, costs $1,199. However, remember that when the Apple II was introduced in 1977, 16K RAM chips cost almost as much as 256K chips cost today. Look for the price of 256K chips to come down—the cost of filling the empty sockets on a RAMworks card will come down with it.

The latest version (3.4) of the RAMworks software also offers users the option of making three other changes to *AppleWorks*. Normally, if you choose to "let the new information replace the old" when saving an *AppleWorks* file, the old file is not actually deleted until the new file has been saved. Because of this feature, the largest file that can be saved on a floppy is 70K—one-half the capacity of the disk. By opting to delete this feature, you can store 140K files on a standard floppy.

If this isn't enough, another possibility is to let the RAMworks software modify *AppleWorks* so that files can be larger than one disk.

The third option available increases the number of records allowed in an *AppleWorks* data base file from the normal 1,350 to 4,283. Using this option, a 1 megabyte RAMworks, and three files, it is feasible to have 10 to 12 thousand records on the *AppleWorks* desktop at the same time.

In addition to the *AppleWorks* modification package, Applied Engineering offers three other software packages that work with RAMworks, but these three cost extra. They include RAMdisk software for ProDOS, DOS 3.3, and Pascal; RAMdisk software for CP/M; and a VisiCalc IIe expander (up to 437K with Advanced VisiCalc and a 512K RAMworks). People who have earlier versions of the *Apple-Works* modification package can get an update by sending in their old disk and $10.

If you own an Apple IIc, all is not lost. Applied Engineering has just announced a card that goes inside a IIc (somehow) and that adds both CP/M capability and up to 512K of memory to that model. This card, which I haven't tested, comes with both the RAMdisk and the *AppleWorks* modification software and costs $449 for the 256K version or $649 for the 512K version (or $159 for CP/M alone—Pascal freaks, add up the price of this card and the $69.95 that Borland International charges for its TurboPascal and compare it to the $250 that Apple charges for Apple Pascal 1.2. You'll find yet another place to both save money and get a higher-quality product).

The technical details of the RAMworks card are that it has room for four banks of memory chips. Two of these banks are on the main card and two are on a piggy-back card that plugs into the main one. Each bank contains sockets for eight RAM chips.

People tend to get confused at this point, because it takes *eight* 64K chips to make 64K *bytes* of memory. The "K" used to describe RAM chips is a count of *bits* on the chips. It takes eight bits to make a byte and eight chips to make a bank of RAM memory (chype?). The RAMworks card can use either 64K or 256K chips, as long as you don't try to mix different sized chips in a single bank.

Thus the minimum memory configuration is one 64K bank. This can be expanded to four 64K banks. Or you can have one or more 256K banks and one or more 64K banks mixed together. If you already have an extended memory card of some other type, you can pull the RAM chips off of it (they're probably the most expensive part of the card anyhow) and put them on a RAMworks card.

Once the RAMworks card is in use, it is invisible to software written for 48K or 64K systems. Software that looks for a 128K Apple will find RAMworks and can interact with it in exactly the same way as with Apple's card.

From a software standpoint, the additional memory on a RAMworks card is organized as extra 64K auxiliary banks of memory. A program can select which 64K bank should appear by writing the desired bank number (0 through 15) to address $C073 (49267). After the bank is selected, a program can use the built-in auxiliary memory softswitches on a IIe to select the various options for reading from or writing to the selected bank. Thus a program can transfer data between main memory and any of the auxiliary banks by itself or by using the Apple IIe's built-in memory-move routines. Data transfers between different 64K banks, however, must be done by using an intermediate move to main memory.

Bank zero contains the 80-column and double hi-res video pages. Video information is always *displayed* from bank zero of RAMworks, even if another 64K bank is active. Applied Engineering has applied for a patent on this feature. However, when *storing* information on the 80-column or double hi-res video pages, bank zero must be the active auxiliary bank, as usual.

The address RAMworks uses for selecting memory banks also triggers the Apple II "paddle stobe", which is used to read the paddles. This means programmers should always insert a 3 millisecond delay before trying to read the paddle/joystick position. This should be standard programming procedure anyhow, since multiple paddle-reads without a delay can create bad reads (for the details on this see *Nibbling at the Game Paddle Port*, by Peter Baum in the October 1984 *Nibble*, page 100).

Unfortunately, the contents of the RAMworks bank register can't be read. Programs must keep track of the current bank number. Applied Engineering recommends that this be done by storing the bank number in reserved locations inside each bank and suggests $FFF0 and $47B be used for this.

The bank register is automatically initialized as zero during power-up, but not after a reset. This means that programs that use other banks should store a zero in the bank register after a reset. Programs that use interrupts must also install an interrupt handler in each bank of memory. When an interrupt occurs on an Apple II, the microprocessor jumps to the address in the interrupt vector at $FFFE. On a 1 megabyte Apple, there are 16 $FFFEs in auxiliary memory, plus 1 in ROM and 1 in RAM in main memory. Thus interrupts get complicated, but they can still work if everything is thought out carefully.

Seeing red in RGB

In North America, there are three different technologies available for displaying Apple II screen output. In this article we'll examine each of them, talk about their advantages and disadvantages, and examine some technical details programmers need to know to use the most expensive of the three, RGB, with double-resolution graphics.

If you already own a television set, the cheapest way to display what's on your Apple's mind is to use a device called an "RF Modulator." This gadget takes the video signal coming out of your Apple and puts it on a "channel" so your TV can receive it. After making all the connections, you can watch your Apple think by tuning in channel 3 or 4 or 33 or whatever.

RF Modulators are available for all models of Apple II. On the IIc, the modulator plugs into the back-panel connector with the color-TV icon. On other models the modulator usually installs inside the case and is wired to a special set of pins called the "Auxiliary Video Output Connector." Modulator prices vary from about $30 to about $60.

The primary disadvantage of using an RF modulator is that 80-column text is unreadable on a television set. Even 40-column text is fuzzy on some sets. Advantages of this technology are a relatively low price, color displays (if you are using a color TV), and the ability to change channels and watch *Laverne and Shirley* reruns when you get discouraged with computing. With a IIc and an RF modulator, you can also route your Apple's sounds through the TV's speaker; the sound quality I've gotten has been disappointing, however.

The composite monitor is the most widely used display technology among Apple II users. In the U.S., an "ordinary" composite monitor is one that accepts a video signal compatible with a standard known as NTSC (National Television Standards Committee). Outside North America other standards, (CCIR, PAL, SECAM) are used. A major advantage of composite monitors is that they can be directly connected to the small round video connector found on the back of all Apple IIs; no additional equipment is necessary. Composite monitors are available in monochrome and full-color versions.

Monochrome monitors have two primary advantages. First, they are relatively cheap—about $100 to about $250. Secondly, they provide the sharpest display available for work involving 80-column text. Their primary disadvantage is that they cannot display full-color graphics. The fashionable display color for monochrome monitors appears to be amber this week, but green and plain old white are also popular.

Full-color composite monitors do color graphics somewhat better than a color television. They have no other known advantages. They are expensive, 80-column text is bad, and you can't change the channel.

The third available technology is RGB (red-green-blue). This is the only system that allows both full-color graphics and readable 80-column text on the same display screen. The price you pay for this technology, however, is high. An RGB monitor costs $400 to $600 and you also have to buy a special RGB adapter to plug into your Apple.

The RGB adapter goes into the connector with the color-TV icon on the IIc, into the auxiliary slot on the IIe, or into slot 7 on a II-Plus. If you are using an Apple memory card in the auxiliary slot of a IIe, you'll need to replace it to use RGB color. The RAMworks card discussed elsewhere in this letter can be adapted to provide RGB signals with the addition of a special piggy-back card that can be added at any time.

(Actually, if this is a piggy-back card, then the card for adding extra memory chips to RAMworks must be a piggy-stomach card, since it goes on the other side—they can both be used at the same time. The whole thing looks like a ham sandwich. You can also get a card into slot 1, but the fit is very tight—the cards touch each other. So maybe you better make that a *grilled* ham sandwich.)

With Applied Engineering's RGB option for RAMworks, incidentally, the color of text displays is switch selectable. You can make your expensive color monitor appear to be a white, green, amber, or blue monochrome monitor when working with text only. However, 80-column RGB text is not as sharp as what monochrome monitor users are used to.

The biggest surprise of using RGB with an Apple II is that double-resolution graphics don't always work. While standard single-resolution Apple graphics always come in loud and clear on an RGB system, double-resolution graphics don't. In order to see *anything* when double-resolution graphics are displayed, you must have *both* an RGB card that supports double-resolution graphics *and* software that turns on the RGB card's double-resolution feature.

And yes, don't fool yourself, there *are* RGB cards around that don't support double-resolution graphics and there *are* commercially available double-resolution graphics programs that don't turn on RGB. So there you go, Bo-Bo; you just spent $800 on an RGB system and sometimes it doesn't even do what a $250 color television can do.

One of the programs that does support RGB double-resolution is *Beagle Graphics*. Mark Simonsen wrote *Beagle Graphics*, and he says Apple-compatible RGB cards are supposed to support at least three modes of double-high-resolution. The *monochrome* mode has a resolution of 560 x 192 pixels (one bit per pixel) and no colors; the *color* mode has a resolution of 140 x 192 pixels (four bits per pixel) with 16 colors; and the *mixed* mode combines these two.

(In mixed mode, the RGB card looks at the high bit of each byte to determine whether it should decipher the bits in that byte as color (high bit on) or monochrome (high bit off). In the other double-resolution modes the high bit of each byte is ignored (it doesn't even matter whether you're talking RGB or NTSC composite). In standard single-resolution graphics, of course, the high bit is a color-selection bit. If you don't know what all this means, don't worry about it.)

Turning the various RGB modes on and off is a matter of peeking or poking at softswitches. The softswitches used to control RGB are the 80-column off and on switches at 49164 and 49165 ($C00C-C00D) and the single-resolution off and on switches at 49246 and 49247 ($C05E-C05F; these also control annunciator 3). These four switches are accessed in different sequences to turn on different RGB double-resolution display modes. The control sequences should be executed each time double-high-resolution graphics are turned on. Execution should occur *after* the standard switches for graphics ($C052, $C057, $C050) have been thrown.

Double-high-resolution programmers note carefully: If you want your software to work on RGB systems, you *must* use the following control sequences to turn on double-high-resolution:

monochrome	color	mixed
80COL off	80COL on	80COL off
SNGL.RES off	SNGL.RES off	SNGL.RES off
SNGL.RES on	SNGL.RES on	SNGL.RES on
SNGL.RES off	SNGL.RES off	80COL on
SNGL.RES on	SNGL.RES on	SNGL.RES off
80COL on	SNGL.RES off	SNGL.RES on
SNGL.RES off		SNGL.RES off

In addition to these three modes, some RGB cards may have additional modes. Sixteen-color 40-column text (using the display page in auxiliary memory for color infomation) and a 160 x 192 mode that has two 16-color pixels per byte (and a straightforward screen-mapping) are two possibilities that have been mentioned, but not yet documented.

Lutus puts editor in public domain

Paul Lutus, author of *Apple Writer*, has placed a rudimentary version of this widely-used word processor in the public domain. The new program is ProDOS-based and is called *Freewriter*. It was recently distributed to Apple user groups by the International Apple Corps on IAC disk 43; many groups have released or are releasing /IAC.43 to their members as a disk of the month.

Freewriter's appearance and command structure are very similar to those of *Apple Writer*. It uses standard ProDOS text files. If you've been looking for a cheap way to peep into text files while using ProDOS, this may be it.

Like *Apple Writer*, *Freewriter* has a status line at the top of the screen that shows how much memory has been used (maximum file size is a little more than 30,000 characters). It uses an 80-column display when run on an Apple IIe with 80-column card or on an Apple IIc with the 80/40 switch in the down position. It uses a 40-column display otherwise.

Commands are included to load and save full or partial files, to find and replace, to jump to the beginning or end of a file, to erase a file from memory, and to embed control characters within a file. The right and left arrows can be used in conjunction with the open-apple key to delete characters and recover them. *Freewriter* has one command heretofore available only in the newest version of *Apple Writer*; it allows you to select a page width from 1 to 240 characters—if a width wider than the display screen is selected, horizontal scrolling keeps the cursor in view. *Freewriter* also has a built-in tutor.

But that's it. Here are some of the things *Freewriter* doesn't have that *Apple Writer* does: case-change mode; find carriage returns or wildcards; the glossary (macro) function; tabbing; disk commands (*Freewriter* has only load, save, and catalog); printing (wait! I'll explain in a minute); word processing language; everything that appears on *Apple Writer's* additional functions menu (control-Q) except quit; replace characters mode; delete word and delete line; and the split-screen function.

While *Freewriter* includes no facility for printing files, a separate Lutus freeware program on /IAC.43, this one written in Basic and called PRINTER, slowly prints text files created with *Freewriter* (or any other program, for that matter). It allows you to enter a page header if desired (including page number), and to choose margins and page length. (/IAC.43 also includes several other free Lutus programs written in Basic for your perusal and enjoyment.)

Freewriter will boot and run on any Apple having at least 64K of memory, but special problems occur on older Apples because they lack some necessary keys. In particular, deletion (which wasn't very good to start with) becomes impossible, since there is no delete key and no open-apple key. Button zero on a joystick or

Writing programs with a word processor

The *only* way to write any but the most simple program is with a word processor. The ability to deal with a program on a larger than line-by-line basis will set your spirit free. In addition, the find and replace feature found in most word processors is extremely useful for determining where in a program certain variables are used and for fixing the inevitable conflicting uses.

For the technique to work, your word processor must be able to save your work as a standard text file. *Apple Writer* and *Freewriter* do this automatically, *AppleWorks* does not. To create a standard text file with *AppleWorks*, pretend you want to *print* the file; when you are asked where you want to print it, select "A text (ASCII) file on disk."

Once you have your program saved, get out of your word processor and into Basic. Then EXEC your file, and Uncle DOS will type the program in for you.

Your file can include both *immediate* and *deferred* commands. (Deferred commands are the ones that begin with a line number, immediate commands stand alone.) I always put an immediate NEW at the beginning of the file, so that if I forget to clear memory before EXECing it will happen automatically. It's also handy to end the file with a SAVE PROGRAM.NAME (followed by a least one Return, more doesn't hurt here), so that the Basic version of the program will be saved to disk automatically, as well.

If you have a program that's already half-written and you want to start using this technique tonight, here's a little exec file that will turn Basic programs into text files. It's a modified version of a program that appears in Apple's DOS programming manuals and works with both DOS 3.3 and ProDOS:

```
0 D$=CHR$(4) : POKE 33,33 : INPUT "FILE NAME? ";F$ : PRINT D$;"OPEN";F$ :
PRINT D$;"CLOSE";F$ : PRINT D$;"DELETE";F$ : PRINT D$;"OPEN";F$ : PRINT
D$;"WRITE";F$ : LIST 1, : PRINT D$;"CLOSE";F$ : TEXT : END

RUN
```

Type this program in on your word processor and save it. You should only press return three times while creating the file, twice after END and once after RUN. When you have the program saved, get back to Basic and load your unfinished symphony. Then EXEC the new file.

It will add a complete Basic program, as line zero, to your program in memory. Then it will run the little one-liner. In response to the "FILE NAME?" question, give a name for the new text file that will hold your program lines. (Note carefully: if you have an extra return following the RUN command, *it* will be used as the filename and you'll die of syntax erroritis.)

Once you have your program in a text file, you can load it directly into *Apple Writer* or *Freewriter*. To get it into *AppleWorks*, don't take the natural route of trying to add a file from disk, but instead select "Make a new file for the Word Processor." Another menu will appear. Select "From a text (ASCII) file" rather than "From scratch," and you'll be ready to edit.

game paddle can be used to take the place of the open-apple key, but it's no fun.

Other problems on older Apple IIs include the impossibility of entering lower-case letters without an enhanced keyboard, a tutorial that is unreadable without a lower-case chip, and the lack of up and down arrow keys (but you can use control-K and control-J instead).

Assembly language programmers should be particularly interested in this disk for two reasons. In addition to *Freewriter*, /IAC.43 also includes the first 14 of the 16 ProDOS technical notes that have been released by Apple. These are additions to and amplifications of the material in the *ProDOS Technical Reference Manual.* If you are interested in writing assembly language programs that run under ProDOS, this material is essential.

The second reason assembly language programmers might want this disk is to disassemble *Freewriter.* I cut my assembly language teeth by taking apart Lutus's *Apple Writer 1.0* several years ago. You can learn a lot by studing a master like Lutus. (In fact, if you'd like to watch a master disassemble a master, get a copy of *Call-A.P.P.L.E. In Depth #4, All About Applewriter IIe,* by Don Lancaster. The book takes you through a step by step disassembly of *Apple Writer IIe.*)

What I learned at MACUL

I spent a couple days near the end of March in Detroit. I was giving speeches about the Apple II at the annual meeting of the Michigan Association for Computer Users in Learning.

It was enlightening to meet people who are using Apples in education and to hear about their trials and successes. And when not giving speeches myself, I was able to sneak into some of the other sessions and pick up some interesting tidbits.

One of the speakers was Sue Talley, whose speech was called "Apple's Commitment to Education." Talley is one of Apple's marketing wizards; she specializes in the education market. She faced some tough questions from the audience about Apple's commitment to keeping future models in the Apple II family compatible with what schools have today.

She responded that computer models change every two to three years while educational institutions tend to update technology on a much slower cycle—every ten years or so. Thus, "changing technology will cause you some problems," she admitted. But the primary market for the Apple II is in schools, and Apple currently holds over half of that market. Consequently, Apple is also committed to compatibility.

"We realize a lot you have software on 5-1/4 inch disks. We won't leave you behind as we change technology," Talley said.

Talley also confirmed that Apple has an Apple II version of its Apple Talk network under development.

MACUL's keynote speaker was David Thornberg of Stanford University. Thornberg invented the Koala Pad and the Muppet Learning Keys, and has written a bunch of books on personal computing. Thornberg also gave several sessions on Logo that I attended.

His Logo sessions were very informative, and confirmed my suspicions that the language goes *Beyond Turtle Graphics* (the title of a new Thornberg book due out this summer). His keynote speech, on the other hand, struck a few sour notes when he attacked Basic and insisted that it should be dropped from school curricula. The standing ovation he got, however, means a lot of people in education agree with him. I don't.

The point that Thornberg makes is that there are better languages for teaching students organized problem solving than Basic. My point is that while learning organized problem solving is important, it is not a star that all computer education must revolve around.

People have got to get away from the idea that there is a single language that is "best" for all situations. Basic, for example, is not well suited for large *institutional* programs written by several different people. Pascal, on the other hand, is not well suited for the short and sweet *personal* programs typical of the software computer users write for themselves. Neither Basic nor Pascal is good for mass market *commercial* software, which is almost all written in assembly language. And assembly language stinks for any program that won't be run at least 10,000 times because it takes so long to write.

And there isn't *any* computer language that gives the average mugwump on the street the number crunching power of a good spreadsheet program, or the data storage power of a good data base program.

Thornberg is well intentioned, but he lives in an ivory tower totally concerned with developing the abstract thinking ability of students. I can appreciate that; the world needs abstract thinkers. But that's not all it needs. Besides, who learns the importance of good organization any better than someone who attempts to write a long program in Basic?

It isn't impossible to write structured Applesoft programs, as was amply demonstrated by JoAnne McVicar and Larry Dove, teachers from the Livonia, Michigan public schools. They did a session on the objectives and course content of the Livonia Public Schools' introductory programming course for high school students.

The course is called "Computer Math 1". The course description goes like this, "Computer Math 1 introduces students to concepts designed to help them become computer literate. Students will use the computer to solve problems in mathematics appropriate to their individual mathematical experience. Students will learn the Basic language using a structured approach to programming." The course was developed by a group of teachers from different Livonia high schools. Besides McVicar and Dove the group included Dan Kinczkowski, Ed Segowski, and Jim Winebrenner, who is the district's computer coordinator.

At the beginning of the school year, Computer Math 1 students are given a disk with a skeleton Applesoft program on it. Here's what the program looks like:

```
200 REM ************      5000 REM *******          10000 REM ***********
210 REM  DICTIONARY       5010 REM  INPUT           10010 REM  SPACE BAR
220 REM ************      5020 REM *******          10020 REM ***********

1000 REM ***********      6000 REM ************     15000 REM *********
1010 REM  MAIN BODY       6010 REM  PROCESSING      15010 REM  SORTING
1020 REM ***********      6020 REM ************     15020 REM *********

2000 REM **************   7000 REM ********         20000 REM *******
2010 REM  INSTRUCTIONS    7010 REM  OUTPUT          20010 REM  MUSIC
2020 REM **************   7020 REM ********         20020 REM *******

3000 REM ************     8000 REM **********       65535 REM student's
3010 REM  INITIALIZE      8010 REM  GRAPHICS                  name embedded
3020 REM  VARIABLES       8020 REM **********                 here
3030 REM ************
                          9000 REM ******
4000 REM **********       9010 REM  DATA
4010 REM  HEADINGS        9020 REM ******
4020 REM **********
```

Each project a student does has to use this skeleton. Many projects, particularly early ones, won't use the whole thing; in that case the student can delete unneeded headings. One thing most students have trouble deleting, however, is the last line of the program. Normally Applesoft doesn't allow line numbers above 63999. McVicar and Dove said they learned this trick from the "Bigliner" program on the Beagle Bros *Utility City* disk.

Students also receive a style sheet that explains in greater detail what finished programs should look like. All programs are to begin with an "identification block" (there's no heading for this, since it is itself the "heading" for the whole program) at line 100. This block uses REM statements to identify the title and author of the program, the date it was last revised, and the name and hour of the class it was written for.

The *dictionary* that follows line 200 is a listing of all the variables used in the program and what they are for. If the program uses functions, they must also be defined here.

The *main body* directs the flow of the program, and usually consists almost entirely of GOSUBS.

The *instructions* section is for code that displays instructions to the program's user. All programs are screen-oriented, so these instructions must always appear on the screen. Skills such as centering, preventing screen scrolling, and other tricks for formatting output are taught as a part of the course.

The *initialization* section is used to dimension arrays, define functions, and initialize all variables. The *headings* section is for placing "permanent" instructions and data on the input and output screens.

The *input* section is for INPUT and READ statements, for checking the validity of input data, and for loading files from disk. The *processing* section is for doing calculations and file manipulations. The *output* section is for placing answers on the screen and for saving files to disk.

The *graphics* section is for code that displays graphics. The *data* section is for all DATA statements. The *space bar* section is for a set of subroutines, provided by the teachers when necessary, that prevent scrolling, provide for screen and graphic dumps to a printer, and include a quit option. The *sort* and *music* sections are also for subroutines usually provided by the teachers. Line numbers after 30000 can be used by students to generate any other subroutines needed as the course develops.

Ask (or tell) Uncle DOS

Enhanced IIe and AE Pro

I have installed the new enhanced IIe chips and have been very satisfied. I know others will not have my experience, but ALL of the standard software I use daily works just fine with the new chips, so compatibility has not been a problem for me. I also have a IIc, and have had no compatibility problems with it either.

Regarding the use of ASCII Express version 4.20 with the new chips, I have had no trouble using it with either the enhanced IIe or the IIc. However, after installing the new chips I did have to run the INSTALL program and change my local console selection (select "L" from the main INSTALL menu). The auto selection under console types (number 0) is what I chose. You might try changing the console type to see if that eliminates your problem. I have been using AE Pro on the IIc for six months and on the enhanced IIe for two weeks and it works great. That does not, however, mean yours will work the same as mine (as we well know).

Although I like the new chips, I really doubt that most people will want to run out and get the upgrade. Unless your application requires the new chips, you probably don't need them—although there is a significant improvement in 80-column mode.

Steve Muncy
Dallas, Texas

Amazing. That works on mine, too. Ok, folks, ***Open-Apple's*** *IIe now uses the enhanced ROMs full-time.*

Dealers bend rules

I was lucky enough to get a IIe enhancement kit from a dealer who let me install the chips myself, so I still have the old chips. I wouldn't buy it without the right to keep the old chips, even if the dealer insisted on installing them himself.

James Patton
Littleton, Colo.

Many dealers are apparently willing to sell the kit (at the "installed" price) without installing it, even though Apple has asked them not to. I highly recommend you buy the kit this way so that you can be sure you have your old chips. The kit comes with installation instructions. You'll need a small flat-blade screwdriver to slip under the old chips so you can gently pry them out. Be careful not to get a chip's pins bent out of line, not to put a chip in backwards, and not to zap a chip with static electricity and you should have no trouble.

Whatever you do, don't pay a dealer extra to install the chips. Apple's suggested retail price is supposed to include installation.

Lower case for DOS 3.3

I enjoyed your article on the IIe enhancement ROMs in the April issue. As you mentioned, it's nice to have Applesoft and the Monitor accept lower case input, which of course also applies to ProDOS. Unfortunately, DOS 3.3 still does not accept lower case. I got tired of getting SYNTAX ERRORs every time I forgot to press the CAPS LOCK key, so I worked out a modification to DOS 3.3 to make it accept lower case. Here it is in the form of an Applesoft program:

```
10 REM Lower case for DOS 3.3 (uses $BCDF)
20 FOR I=0 TO 16 : READ J : POKE 48351+I,J : NEXT
30 POKE 41369,76 : POKE 41370,223 : POKE 41371,188
40 DATA 201,224,144,2,41,223,201,141,240,6,232,142,
        93,170,201,172,96
```

I suggest booting with a DOS 3.3 System Master disk to be sure you have a clean copy of DOS before running this program.

J. Morris Prosser
Pebble Beach, Calif.

Your program sticks the lower-case conversion routine inside a normally empty space inside DOS at $BCDF. Many other programs have been designed to use that space as well.

I, too, am tired of CAPS LOCK, but I use $BCDF for something else. I looked around for some other place to stick your routine, but there aren't any left that haven't already been used for something. Then I decided to gulp great big and get rid of something close to my heart that I actually haven't used in a long time, Integer Basic. Here's a version of your routine that should work with even highly-modified copies of DOS 3.3:

```
10 REM Lower case for DOS 3.3 (deletes Integer Basic)

15 POKE 40268,121 : REM make INT do FP
20 FOR I=0 TO 9 : READ J : POKE 40290+I,J : NEXT
30 POKE 41374,32 : POKE 41375,98 : POKE 41376,157
40 DATA 142,93,170,201,224,144,2,41,223,96
```

Of course, there are other ways to solve this problem, too…

Old bugeyes writes

When I read my *Basic Programming with ProDOS* manual, I saw two things that almost convinced me to abandon DOS 3.3…

1. Page 154 and 206: ProDOS makes an HGR or HGR2 command protect the hi-res pages so your Applesoft program and variables won't overwrite them.
2. Page 207: ProDOS lets Applesofts INPUT statement accept anything, including commas and colons.

Apparently the writer of this manual has been sitting on his head too long. Do you have any inside info about this?

Incidentally, a recent reader asked you about how to make his IIe accept lower case keyboard commands. One solution, as you stated, is a IIe enhancement kit. Another solution, with many more benefits, is GPLE.

While we're on the subject, try this apparently illegal statement on your IIc or enhanced IIe:

```
print 2~2
```

("~", as it turns out, is a lower case "^".)

Bert Kersey
Beagle Bros, Inc.

The two ProDOS features you mention were included in early versions of Basic.system, but were removed because of intractable bugs. Somehow, they didn't get removed from the manual, however. Speaking of bugs,…

Wrong number

The phone number listed for Sunset Software last month (page 30) isn't right. Do you have a correction?

Bob Leedom
Glenwood, Md.

Indeed, I seem to have copied the area code down wrong. It should be 213-476-0245.

HUMANTEXT errors

I have just installed the enhancement on my IIe and on entering your HUMANTEXT DEMO from the April issue (page 27) I kept getting errors in lines 110 and 120. I changed them to read as follows:

```
110 M$=CHR$(27) : T$=CHR$(15) : MT$=M$+T$
120 N$=CHR$(24) : R$=CHR$(14) : RN$=N$+R$
```

Even placing a semicolon at the end of the lines caused an error. Why?

By the way, I was not able to configure the Sider with a MicroSoft IIe Premium card in slot 3. CP/M will not work and I've seen some strange sights with Apple Pascal.

Warren L. Posey
Santa Ana, Calif.

I found those same bugs in the HUMANTEXT DEMO when I tested it, but somehow the corrections never got into the file I sent to the printer. Here is how the lines appeared in the April issue:

```
110 MT$=CHR$(27);CHR$(15); : REM turn on mousetext
120 NR$=CHR$(24);CHR$(14); : REM turn off mousetext
```

The semicolons are the problem. The first semicolon in each line should be a plus sign; the second semicolon in each line should be removed. This is a common error, at least around here. The confusion stems from the use of semicolons in PRINT statements. If these lines said PRINT instead of MT$= and NR$=, the semicolons would be correct. Outside of PRINT statements, however, you must connect strings with the plus sign, not with the semicolon. (Alternatively, plus signs are legal for connecting strings inside PRINT statements, but can't be used at the end to suppress a carriage return.)

I haven't tested the Sider with either CP/M or Pascal. Yours is the first report of problems with either one that I've heard, however, and I've seen many reviews lately. The Sider manual does say you should put the CP/M card in slot 4.

To flip or not to flip

I have a question for you that might be of interest to other readers: Is the practice of flipping over a floppy disk and using the other side as an economy measure a good habit?

Benjamin Day
San Marino, Calif.

I was hoping nobody would ever ask. No matter how I answer, half of you will argue that I'm dead wrong. At least you didn't ask about copy protection.

I've been using the flip side of disks for a couple of years now. I haven't experienced any problems. Over the course of time and many, many disks I've run into a few bad ones, but they were bad right out *of the box and they were bad on the side you were supposed to use. It appears to me that if one side works, the other side will work too.*

I got started at this by stealing a one-hole paper punch out of my daughter's pencil box. All you have to do is punch a notch where there ought to be one and put the disk in the drive upside down.

There are a few **theoretical** *reasons why this should cause problems. I think that the reason it doesn't is that disks fill up with data long before the problems start. Estimate how many hours your most-used disk has actually spent spinning inside a drive. I don't think I have a disk that has spun for more than a couple hours. The kinds of programs I use always load a whole file into memory in a few seconds and save them back in a few seconds.*

If you have a program that really makes disks spin—the kind of program that works with disk-based data rather than data loaded into memory—it might be better not to flip the disks you use with that program. Otherwise, I can't see any problems.

Seeing as printers see

I neglected to include a joke on my subscription postcard, so here:

"How many programmers does it take to change a lightbulb?"

"None, that's a hardware problem."

Here's a brief note about the discovery of an undocumented feature of Apple's Imagewriter printer. The manual describes a printer selftest, triggered by holding in the "form feed" button while powering up. What is not described is the hexadecimal dump feature. If one holds in the "line feed" button while powering up, the printer will will print the hexadecimal codes for all characters it receives instead of doing normal printing. This feature is useful for debugging sequences of printer control codes, etc.

Incidentally, this is a standard feature of Epson FX printers and C. Itoh 8510 printers. The fact that the Imagewriter is a C. Itoh with modified firmware prompted me to try this out.

Bernard Goodman
Cambridge, Mass.

I immediately tried your tip on an Apple Dot Matrix Printer and an Apple Scribe and it works on both of those, too. (On the Scribe you have to hold in on "select" while powering up because "line feed" and "form feed" are the same, self-test generating, button.)

I can think of several uses for this feature in addition to debugging printer control sequences. How about using it to figure out the codes graphic dump programs are sending to a printer? With a little work, you could even put a serial printer where a serial modem is supposed to be and debug modem-command sequences.

Text file splitter

I've got an early word processor (*Magic Window II*) which has the drawback of not being able to read a text file that is larger than memory. Because of this, when I create large text files (logging from a database), I can only view the first 12 pages, or so. My question is twofold; first, is there a way to "split" a text file into more managable bytes? Second, if I happen to shell out for a Z-80 board and start running XMODEM for file transfer, does using that protocol also create text files which you then EXEC, or does it create some other type of file?

Maybe the whole problem is my word processing program and I should just buy (ouch!) a new one with better features?

Bill Curtis
BBZ618

Using the CP/M program XMODEM for file transfer will get you a bunch of CP/M files. You'll have to convert the CP/M files to DOS 3.3 before you can EXEC them. Most CP/M cards come with a disk utility that will do this.

Here's a simple program that will (very slowly) split a text file into more managable pieces:

```
10 REM *** TEXT FILE SPLITTER ***

100 D$=CHR$(4)
110 NFL=5000 : REM new file length, in bytes
120 IF PEEK(978)=157 THEN DOS=3.3 : GOTO 200
130 IF PEEK(978)<>190 THEN PRINT
      "ACTIVE DOS NOT RECOGNIZED." : END

200 HOME : VTAB 10
210 PRINT "THIS PROGRAM SPLITS LONG TEXT FILES"
220 PRINT "   INTO SMALLER PIECES. WHAT'S THE"
230 PRINT "   NAME OF THE TEXT FILE YOU WANT"
240 PRINT "   TO SPLIT?"
250 PRINT
260 INPUT F$
270 IF LEN(F$)=0 THEN END

300 PRINT D$;"OPEN";F$
310 ONERR GOTO 500
320 PRINT : PRINT
330 PRINT "OK. NOW THIS WILL TAKE SOME TIME..."
340 PRINT "  (ABOUT 20 SECONDS PER SECTOR)"
350 PRINT

400 FOR N=1 TO 100
410 : PRINT "NOW WORKING ON ";F$;".";N
420 : PRINT D$;"OPEN";F$;".";N
430 : FOR I=1 TO NFL
435 : : IF DOS=3.3 THEN POKE 43602,0
440 : : PRINT D$;"READ";F$
445 : : GET A$
450 : : IF DOS=3.3 THEN POKE 43602,0
455 : : PRINT D$;"WRITE";F$;".";N
460 : : PRINT A$;
470 : NEXT I
475 : IF DOS=3.3 THEN POKE 43602,0
480 : PRINT D$;"CLOSE";F$;".";N
490 NEXT N

500 PRINT D$;"CLOSE"
510 IF PEEK(222) = 5 THEN 530
520 PRINT "ERROR #"; PEEK(222);" IN LINE ";
      PEEK(218) + PEEK(219)*256
530 POKE 216,0 : END
```

Feel free to change the value of NFL in line 110 to whatever is suitable for your word processor. This program doesn't check the contents of the file, it just splits after the amount of characters you specify with NFL have been processed, so you can expect the splits to come in mid-sentence or mid-program line. You should be able to use your word processor to fix such stuff, however.

Be prepared to go watch TV or put your kids to bed while this program is running. It takes about 20 seconds per sector. Your computer will appear to be locked up for most of that 20 seconds, but then your disk drive will come on to reassure you everything is working fine.

The POKE 43602,0 in lines 435 and 450 is a "brute force carriage return" for DOS 3.3. Notice that the FOR loop causes the READ command in line 440 to follow the PRINT in line 460. But the PRINT ends with a return-suppressing semicolon. Uncle DOS will not respond to the READ command unless a return preceeds it. But if you print a return, it will be written to the new file (we would then have a return after every character). The forceful, decisive way to get around this problem is to cheat and reset the carriage-return flag inside DOS by hand. This is what the POKE does.

Likewise, the WRITE command in line 455 will never be recognized unless it is preceeded by a return because of the GET in line 445. (DOS commands always cause trouble after GET.) And the CLOSE command in line 480 needs a returnless return, as well.

The POKE 216,0 in line 530 simply turns off the ONERR GOTO.

POKE 1403 problems

I've discovered a bug involving the *POKE 1403* statement to position the cursor horizontally on the 80-column IIe. [*POKE 1403,X-1* does what *HTAB X* is supposed to do but doesn't on the original IIe.] For some unknown mystic reason, the first (and only the first) *POKE 1403* after a DOS command, including PR#3, is ignored. If you know why this is or how to avoid it, I'd enjoy reading about it.

My own solution is to do a dummy *VTAB Y : POKE 1403,X* to an unused screen location after each DOS operation. Clumsy, but it works.

Mike Fredericks
Littleton, Colo.

Thanks for pointing this out. Lots of people report trouble with VTAB, HTAB, *and* POKE 1403. *Maybe this is the primary problem.*

I don't know why, but another solution seems to be to PRINT *a return after every DOS command. For example,* PRINT D$;"PR#3" : PRINT.

Changes coming to $C020

The Apple II Group is seeking ways to expand the capabilities of our Apple II line. We would like to do this with a minimum of impact on existing products. We have determined that our future needs will require that we change the function of a series of soft switches. In the past the locations $C020 to $C02F (only $C020 was documented) were used for writing to the cassette port. We have decided that the functions of these locations need to change. We are letting you know of these changes now so that you can make needed changes to your software as part of your normal product updates.

In the future, if you read from or write to these locations the current ROM configuration may be banked out. This means the loss of Applesoft, the Monitor ROM and 80-column ROM. It is very important then that your products stop using these locations.

Very little software uses the cassette port to actually write to the cassette. However, some software does write to or read from this port as a way of maintaining program speed when sound is turned off. In these cases sound is turned off by redirecting the poke to the speaker toggle to a poke to the cassette port. Such programs would have to be changed to poke to an internal location or a known "safe" ROM or RAM location.

Apple II Technical Support Group
Cupertino, Calif.

By incrementing the high byte of speaker address you would make it point to $C130, an address within the ROM space of the card in slot 1—one of the few memory areas that is always guaranteed to be "safe."

ProDOS bugs

Here are some bugs that have been discovered in ProDOS that you might want to mention to your readers:

1. In Basic.system 1.1, if you BSAVE over an existing file the old load address and length are retained. I have reverted to 1.0, which does not have this problem, but use the PRODOS 1.1.1 kernel.

2. CONVERT applied to ProDOS —>DOS 3.3 on a file requiring more than one track/sector list (i.e. a file more than 61 blocks long) does not properly set up the track/sector list link or offset to subsequent track/sector lists. It takes a knowledgeable person and a disk zap to make such a file readable.

Glen Bredon
Princeton, N.J.

A representative of Apple's Technical Support Group says the BSAVE problem is a true bug that will be fixed soon. Apparently there are also a couple of very exotic bugs in Basic.system 1.1 that Apple's programmers would like to figure out and fix before putting out another release. You can overcome the BSAVE problem for now by deleting a binary file before saving—much as is done with text files.

The CONVERT problem is quite interesting. Combine it with the problem described in the next letter relating to the IIc System Utilities and you'll find there is currently **no way** *to convert long text files from ProDOS to DOS 3.3.*

Mousetext apologist surfaces

Dawgonnit, with all the neat stuff in your newsletter, it's almost impossible not to write back with a few thousand or so comments.

Perhaps I can add to the rodent discussion. Maybe the reason Apple put the mousetext characters where they did was so that assembly language programmers who desired to store inverse characters directly on the screen could use the original inverse capital set in the $0-$1F range. This retains compatibility with older Apples. Looking at it this way, the alternate character set replaces flashing characters with *two* sets of special characters, mousetext at $40-$5F and inverse lower-case at $60-$7F.

Keep in mind that the ASCII character set and the screen display character set are two independent and unrelated (execpt by sequence) groups of characters. And just for fun, note that the two groups of "normal" caps in the screen display set are not the same. Those (control) characters in the $80-$9F range cannot be traced over using the arrow key. POKE 1025,128 and see!

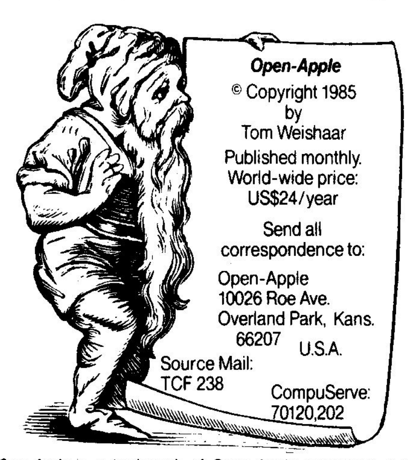
There are substantial differences between the enhanced IIe Monitor and the IIc Monitor. The ability to enter ASCII characters using the apostrophe is great, as is the mini-assembler for quick and dirty stuff. For those who may be interested, Apple has published identity bytes that can be checked to determine which Monitor (and thus which machine) a program is running on, as shown in the following table:

	$FBB3	$FBC0	$FB1E
Apple II	38		
Apple II-Plus	EA		AD
Apple III (in emulation mode)	EA		8A
Apple IIe (original)	06	EA	
Apple IIe (enhanced)	06	E0	
Apple IIc	06	00	

The Apple IIc System Utilities disk is really el-neato; it does everything that FILER and CONVERT should, but don't do. You don't have to tell it volume names or identify a disk as 3.3 or ProDOS, etc. Unfortunately, it has a couple of major bugs. From ProDOS to DOS, it will fill all sectors of a text file after the 46th with garbage instead of data, and from 3.3 to ProDOS, a long binary file will cause an unrecoverable system crash.

Val Golding
El Cajon, Calif.

You make a good point about mousetext, but in the wisdom of hindsight I think Apple's wizards should have put it in the $80-$9F range where the useless normal capitals are. Your point about not being able to trace over that range would even be an advantage, since it makes no sense to trace over mousetext characters anyhow.


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

Lam a'la Ruth

In February (page 12) you showed readers how to use the Lam technique for entering Monitor commands from within Basic programs. Some time ago I wrote a Mockingboard speech editor in which I wanted to enter strings of phonemes in the form of hexadecimal numbers. I planned to use the Lam routine to put the numbers into memory. However, repetitious use of Lam to handle lengthy Monitor commands turned into a serious drawback because of the routine's slow execution speed.

The routine's sluggishness comes from its use of the MID$ function to extract characters from C$, one at a time, and poke them into the keyboard input buffer. Applesoft string operations are notoriously slow, and including them in a loop is one of the best ways I've found to waste time.

I circumvented this problem by using the normal Applesoft version of the Lam routine to create a machine language version of itself. This way the slow Lam is executed only once, and all subsequent Lam requirements are handled by Lam a'la Ruth. The speed improvement is phenomenal!

The C$ string in the following program creates Lam a'la Ruth. It uses Monitor commands to install a machine language routine at $300 and to connect the routine to the Ampersand hook. After C$ has been installed using the standard Lam technique (line 110), this program installs it a second time using the new technique (line 130). Beeps will give you a clear idea of the speed improvement.

```
10  REM *** LAM A'LA RUTH ***

100  BELL$=CHR$ (7)

110  C$="300:A5 71 48  A5 72 48  20 E3 DF  A5 83 85
             AB A5 84  85 AC A9  00 85 71  A9 02 85
             72 20 D4  E5 A0 00  B9 00 02  30 0A F0
             08 09 80  99 00 02  C8 D0 F1  A9 8D A0
             00 91 71  84 48 20  70 FF 68  85 72 68
             85 71 60  N 3F5:4C 00 03  N D9C6G"

200  PRINT BELL$
210  FOR I=1 TO LEN(C$) : POKE 511+I,ASC(MID$(C$,I))
          +128 : NEXT : POKE 72,0 : CALL -144
220  PRINT BELL$
230  & C$
240  PRINT BELL$
```

Here's a listing of the machine code installed at $300. Note the liberal use of Applesoft's own internal routines to accomplish the task. Not everything in Applesoft is slow. Instead of picking C$ apart piece by piece, the routine moves it en masse into the buffer. This is where all that time gets saved. Since Applesoft stores strings in low-value ASCII (0-127) and the Monitor expects its commands in high-value ASCII (128-255), extra code (from $31C to $32C) is used to make the conversion.

```
0300: A5 71     LDA $71     Save the temporary string
0302: 48        PHA           pointers on the stack.
0303: A5 72     LDA $72
0305: 48        PHA

0306: 20 E3 DF  JSR $DFE3   (PTRGET) Find specified
                              string and point to it.

0309: A5 83     LDA $83     Transfer string pointer
030B: 85 AB     STA $AB       from where PTRGET put
030D: A5 84     LDA $84       it to where MOVINS can
030F: 85 AC     STA $AC       find it.

0311: A9 00     LDA #$00    Tell MOVINS we want to
0313: 85 71     STA $71       move it to page 2
0315: A9 02     LDA #$02      (the keyboard input
0317: 85 72     STA $72       buffer).

0319: 20 D4 E5  JSR $E5D4   (MOVINS) Do the move.

031C: A0 00     LDY #$00      Examine relocated string.
031E: B9 00 02  LDA $0200,Y
0321: 30 0A     BMI $032D     If low-value ASCII
0323: F0 08     BEQ $032D       or null, we're done.
0325: 09 80     ORA #$80      Convert low-value ASCII
0327: 99 00 02  STA $0200,Y     to high-value
032A: C8        INY           Continue till whole string
032B: D0 F1     BNE $031E       or all of page 2 is done.

032D: A9 8D     LDA #$8D      Place $8D (return) at the
032F: A0 00     LDY #$00        end of string ($71 points
0331: 91 71     STA ($71),Y     there after MOVINS).

0333: 84 48     STY $48       Poke 72,0
0335: 20 70 FF  JSR $FF70     Call -144

0338: 68        PLA
0339: 85 72     STA $72     Restore original temporary str
033B: 68        PLA           pointer before returning to
033C: 85 71     STA $71       Applesoft.
033E: 60        RTS
```

Once the standard Lam has installed Lam a'la Ruth, your subsequent Lam applications need only define C$ and then & C$. Better fasten your seat belt!

Clay Ruth
Dyer, Indiana

Of the several methods available for entering machine language programs from within Basic programs (POKE,POKE,POKE; READ DATA POKE; LAM), I've always preferred the Lam technique because it is very efficient in terms of keystrokes. The other methods did have speed advantages, however, until now. Thanks for this improvement of an old technique.