Open-Apple™

November 1987
Vol. 3, No. 10

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

Releasing the power to everyone.

Reality and Apple's vision

The Apple II was introduced in 1977 at the first West Coast Computer Faire. The show opened on April 16 in the San Francisco Civic Auditorium. Ten years, four million computers, and a few months later, the same auditorium was the site of September's AppleFest. Apple was distributing buttons that read "Ten Years Strong." Finally, after ten years, no one was predicting the imminent death of the Apple II.

AppleFest's opening keynote speech was given by Del Yocam, Executive Vice President and Chief Operating Officer of Apple. Yocam's speech was titled "Apple Home Movies" and was dedicated to celebrating the tenth birthday of the Apple II. Yocam's presentation included clips from old Apple commercials and videos.

"You know," Yocam said, "the Apple II might have lived and died just another gadget if it hadn't been for two circumstances. First, people could learn to program on an Apple II using BASIC, a relatively simple programming language, without having to load the language into memory....And second, the floppy disk drive developed by Woz in 1978 assured the Apple II a long life ....The Disk II and its disk controller card opened the floodgates for Apple II software. And it was the software that brought millions of new customers to the Apple II.

"The rest of Apple II history is largely a history built on software that we didn't even develop. That is what I think is the most remarkable thing about our success....The success of the early software, like *VisiCalc* and *Apple Writer*, demonstrated to programmers how lucrative programming could be. The floodgates were open. Programmers developed, customers bought Apple II's, and the circle went 'round....

"In the eight years I've been at Apple, we've built five generations of Apple IIs....In all that time, there was never a doubt on our part that there would be a next generation Apple II. There are over 4 million Apple IIs in the world today. But you can still run Apple II software from 1977 on a IIgs. In my mind our greatest achievement is to advance the technology and take our customers with us.

"The Apple II exemplifies our dream to bring the power of computers to individuals....We've spent the last ten years making the Apple II better, all the while fulfilling our dream of building great, truly personal computers. ...Without a doubt, the Apple II is advancing....From Apple's point of view I can tell you that in ten years the technology will be to today's technology what the IIgs technology is to the Apple I Woz built for the Homebrew Computer Club. We'll advance the Apple II in networking and communications, in performance, in new types of peripherals, and in system software and programming tools....We are working on multi-media, CD-ROM, and videodisc technology."

At AppleFest I was enthused by Yocam's enthusiasm for the Apple II. I was struck by his vision of what the Apple II will become in the next 10 years. Back home, however, reviewing the text of Yocam's speech, I awoke to the fact that Yocam had celebrated the past and future of the Apple II without once mentioning AppleWorks.

AppleWorks was introduced in late April 1984, about three months after the Macintosh. The program was originally written for the Apple III by Robert Lissner, an independent developer who had previously done *Quick File* for Apple.

While AppleWorks was under development, Steve Jobs was flying a pirate flag over the Macintosh building and outfitting that machine with the famous interface developed by Xerox researchers. Over on the Apple II side of the street, Apple's own researchers were using human subjects sitting at Apple keyboards to refine a user interface based on cursor pointing, return key selection, and a "desktop" with a "clipboard" for transferring data between files. Lissner incorporated the results of this research into AppleWorks.

However, according to popular legend, after AppleWorks was finished Apple attempted to find an independent company to publish the program. Apple itself had two major problems with AppleWorks.

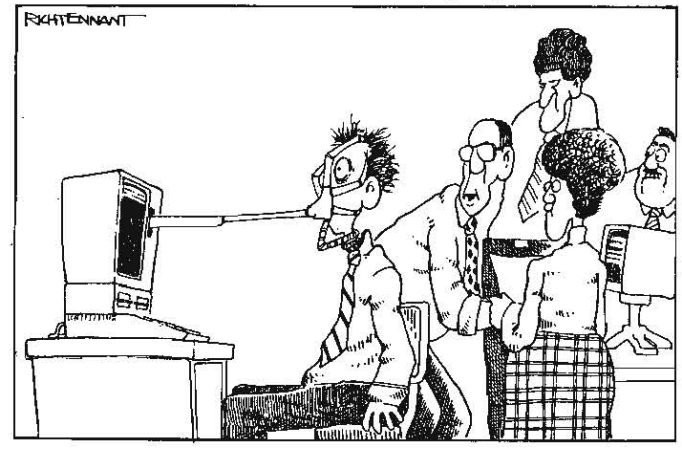
First, Apple was trying very hard to get the big MS-DOS developers to work with the Macintosh. One of the reasons these developers gave for their reluctance to work on the Mac was their fear that Apple itself would compete with them—Apple, obviously, had tremendous advantages in terms of distribution and access to inside information. Apple had a reputation for developing applications software for its machines that would kill the market for similar software—*Apple Writer* (which was at the top of the Apple II software charts at the time) and a complete set of applications software for the Lisa being major examples. Powerful voices inside Apple wanted the company to get out of the applications software business.

Secondly, Apple was pouring all of its energy into promoting the graphics-based Xerox interface used on the Macintosh. Among the people who were to decide how Apple's marketing dollars would be spent, there was little hope for, and less interest in, the text-based interface used by AppleWorks.

But, the legend goes, no independent company with the resources and willingness to publish AppleWorks could be found. And on a warm spring evening, in a joyous moment of weakness that followed months of intense attention to its new Macintosh and Apple IIc babies, Apple let AppleWorks slip out the door.

Apple's punishment for its indiscretion was immediate—within six weeks its illegitimate child sat at the top of the Apple II best-seller lists. AppleWorks achieved this without the benefits of a mother's love—it succeeded in spite of, not because of, Apple's meager marketing efforts in its behalf.

Since AppleWorks was released, for example, Apple has run 26 pages of ads in *A+* magazine. The word "AppleWorks" appears in those ads exactly zero times. Four of the ads show screen shots of AppleWorks. Take a look at Apple's IIgs ad in the September 1987 *A+* if you have one handy (pages 33-39). AppleWorks appears in the gutter between the pages and is the only one of the 23 programs shown that isn't mentioned by name. This is typical of the treatment Apple's bastard child gets from its mother. Yocam didn't mention it or Lissner in his birthday speech, and John Sculley, Apple's president, doesn't mention it or Lissner in his new book, *Odyssey*.



"ALRIGHT, STEADY EVERYONE. MARGO, GO OVER TO TOM'S COMPUTER AND PRESS 'ESCAPE',...VERY CAREFULLY!"

Before the end of 1984, the year of the Macintosh, a software product for an Apple computer had dislodged the almighty *Lotus 1-2-3* from the number 1 position in monthly retail software sales. One would expect no end to the ballyhoo. But the Lotus-killer was AppleWorks, and corporate Apple celebrated the triumph with silence.

During 1985, 1986, and 1987, corporate Apple's reaction to its illegitimate child has been to continue to pretend that it doesn't exist. AppleWorks, in turn, acts like a child of Satan as far as Apple is concerned. Not only does the program's success heap scorn on the Xerox interface, AppleWorks eats third-party software alive. In Cupertino, the fact that since mid-1984 Apple itself has earned a large proportion of the profits available in the Apple II software market has been a matter of concern rather than a matter of glory.

It's time for Apple to stop pretending, to modify its vision, and to admit to the reality of what the Apple II has become since the Macintosh was introduced. In the three years since January 1984 more than 2 million Apple IIs have been produced and sold—well over half of all Apple IIs in existence. Just as *VisiCalc* made the Apple II-Plus, just as *Lotus 1-2-3* made the IBM-PC, just as desktop publishing has made the Macintosh, AppleWorks made the IIe and IIc.

Reality is that AppleWorks has become the essential characteristic, the defining characteristic, of the Apple II family. They laughed when AppleWorks sat down to play integrated software on an 8-bit computer, but *nowhere* has integrated software met near the success that it has on the Apple II. AppleWork's speed, ease of use, and seamless integration have set standards for all Apple IIs of the future.

Reality is that more humans have learned the AppleWorks interface during the last three and half years than have learned the Xerox interface—even though the Macintosh had a three-month head start. Apple's insistence that the Xerox interface is the only interface in Apple's future is as stupid as saying that the only legitimate music is rock and roll or that the only legitimate literary form is the romantic novel. General Motors got to the straits it is in today by removing the distinctions between its automotive families, not by letting a Chevrolet be a Chevrolet and a Pontiac be a Pontiac.

Reality is that the productivity area (word processors, data bases, spreadsheets, and similar programs) of the Apple II software market is a shambles. Apple has not only taken all the profit out of this market for three and a half years—*it has put nothing back in.* Pretending and wishing that it wasn't happening, which has been Apple's primary reaction to date, has us on the edge of a disaster. (A secondary reaction has been spinning off a Macintosh software company and attempting to sweep AppleWorks under its skirts.) Only one established vendor of productivity software, Word Perfect, attended AppleFest. Where will future Apple II productivity software come from? At a dinner for developers the night before AppleFest, Sculley said that the Apple II software market is the only one left in which small, independent developers aren't totally dominated by larger companies. That's true and, in general, that's good. But if Apple couldn't find a company strong enough to publish AppleWorks in 1984, before it spent three years soaking up all the profits in this market, how could there be one now? Sculley thinks there are companies in the Apple II market with the resources to develop a product such as *Hypercard*, which took a team of the brightest stars at Apple three years to develop for the Macintosh. But Sculley said he's looking to independent developers to provide that product for the Apple II. Given the amount of money earned by Apple II developers during the last three years, the idea is ludicrous.

Reality is that homes are the final large market for personal computers. As the number of computers in schools and businesses mounts, we are approaching a threshold across which millions of people will decide to purchase computers for their homes. The Apple II's dominance in schools will put pressure on parents to buy Apples. However, the dominance of MS-DOS in businesses will put pressure on parents to buy clones. Apple is giving a large share of the home market away to the clones by not marketing the Apple II as a machine capable of giving both kids *and adults* the power to be their best at home. For example, on October 12 Apple sponsored a network television celebration of the bicentennial of the U.S. Constitution. Of the twelve commercials shown, eight had to do with the Apple IIgs and kids or education, two with Apple itself, and one with Apple IIs and the disabled. The sixth commercial shown was titled "The Home Office." It alone promoted the idea of computer use by adults and it alone promoted the Macintosh. Taken as a whole, Apple's message was that the Apple II isn't suitable for use at home by adults. This isn't a message that will maximize Apple's share of the home market, nor is it a message that reflects the reality of hundreds of thousands of Apple IIs in homes right now giving adults the power to be their best.

Apple will never acknowledge the existence of its illegitimate child or begin to show that child love without first bringing its corporate vision in line with reality. What Apple must incorporate into its vision is what Apple II users realized months ago—that AppleWorks has shifted the distinction between "systems software" and "applications software." AppleWorks is the *systems software* that Apple II users want built into their machines—AppleWorks is the "*platform*" that applications developers want to build from.

Here are the ideas I would champion if I worked for Apple.

First, corporate Apple should acknowledge Robert Lissner's contribution to the Apple II. He should be an Apple Fellow, like Bill Atkinson, developer of *MacPaint* and *Hypercard*; and Alan Kay, developer of the Xerox icons.

Second, corporate Apple should acknowledge what Apple II users already realize, that the "operating system" of the Apple II ends on the AppleWorks desktop. AppleWorks should be in the box with current Apple IIs and should be built into the ROM of new Apple IIs (with plenty of RAM-based hooks for independent developers to get their teeth into).

For example, a developer of application languages should be able to allow users to write programs with the AppleWorks word processor. When the user is ready to run the program, open-apple-K could compile and execute the program. The program should be able to take control of the screen, to request AppleWorks desktop memory, to access files on the desktop and on disk, to use all other operating system tools, and so on.

For example, a developer of communications software should be able to add a communications capability to the AppleWorks word processor. A developer of graphics software should be able to write a program that is activated whenever a graphics file is added to the AppleWorks desktop. A developer of spelling checkers should be able to write a program that works inside AppleWorks. And these developers should be able to do these things without worrying that Apple will add those features to a future version of AppleWorks and cut their market out from under them.

(For additional information on what "applications software" is if AppleWorks itself is "system software," take a look at the programs in Beagle Bros' new Time Out Series. Contrary to the popular expectation that IIgs products would dominate AppleFest, most of the people I talked to said they considered the Time Out Series to be the big hit of the show.)

Apple is reportedly investing a large part of its research and development funds in system software technology. A fair portion of that money should go into a system software "platform" based on AppleWorks that defines an Apple II, rather than one that defines a Macintosh.

Third, corporate Apple should acknowledge the havoc its illegitimate child has caused in the Apple II software market. Apple should reinvest the dollars it has made from AppleWorks in a "Marshall Plan" for Apple II software developers. The plan should be directed at small one- and two-programmer companies that are committed to the concept of Apple II Forever, rather than at large companies that are primarily Macintosh and MS-DOS developers. The plan must be designed as an *investment in the future of the Apple II* rather than as a profit center.

The plan should include strengthening Apple II Developer Technical Support to the point that it can provide training by correspondence (rather than by a trip to California that is too expensive in both dollars and time). Developer support should also provide electronic forums for the discussion and implementation of software standards, an interface between software developers and Apple's own hardware and systems software engineers, and a continuation of its current technical help by electronic mail.

In addition to technical help, Apple II developers need marketing help. Presently this kind of help goes only to Apple II developers who share Apple's vision of the Apple II as a toy Macintosh. This help has to be extended to all Apple II developers, no matter what their vision. In the Apple II kingdom, it has always been developers and customers who have, in the end, determined the direction of the Apple II. Apple's help should be available to all.

Since 1984, Apple's vision of the Apple II and the reality of the Apple II have been getting farther and farther apart. It's time for Apple to blow away the Macintosh smoke and look at what the Apple II has actually become. The reality is a machine defined by AppleWorks. The good part of that reality is that the Apple II has developed its own identity, distinct from Macintosh and MS-DOS machines. The future of the Apple II will be unlimited if corporate Apple builds on reality by making AppleWorks part of the Apple II "system software platform" and markets the power of that reality to adults with the enthusiasm it has heretofore generated only for the Macintosh. The bad part of that reality is that Yocam's observations about why the Apple II was a success are no longer true. Few software companies besides Apple itself have profited from Apple II development in the last three years. If Apple is serious about producing advanced Apple IIs ten years from now, it should invest its AppleWorks profits now in rebuilding the Apple II software industry.

Control-I(nterface) S(tandards), continued

"Rinnnng...rinnng...Hey you, get up there. This is Oscar the Grouch and I'm telling you it's time to get up. And don't forget to wind the clock so I can wake you up again tomorrow....Rinnnng...Rinnng...Hey you...."

Early feedback on last month's lead article indicate it was a sleeper, especially for those of you among our readers who don't write programs. I hope you'll understand that I occasionally have to resort to articles like that to make your lives easier in the future. Based on the mail we get, making a printer do what the salesman said it would still seems to be one of the primary problems in the Apple kingdom. Most of these problems are rooted in a lack of standards for printers and printer interface cards.

Open-Apple's readers extend from those of you who don't program, up thorough novice and intermediate programmers, up through commercial hardware and software developers, and right into the labs where Apple's engineers are designing future Apple IIs. We are all, ultimately, Apple II users. Things will go smoother for all of us if the programmers, developers, and engineers among us can agree on, understand, and follow a few more standards than we have in the past. ***Open-Apple*** is in no position to promulgate Apple II standards, but I do want to provide a forum for raising issues, for uncovering problems, and for providing pressure for better Apple II standards. That's part of what you're paying for when you subscribe to ***Open-Apple*** (and a little nap now and then is healthy, they say).

At any rate, this month's quest should be more interesting to most of you. Instead of looking at interface cards from the programmer's point of view, we're going to look at them from the user's point of view. We're going to investigate in detail the *user commands* that control Apple's three most popular serial interface "cards" or "ports." There are significant differences in what these commands do and how they work. Detailing these differences here in ***Open-Apple*** should help users get the most out of their interface cards, should help programmers and developers understand how to get their programs to work on any Apple II (even yours), and should help Apple's engineers to stay consistent.

The three interfaces we'll be examining are the Super Serial Card, which is used in II-Pluses and IIes, and the serial ports on the Apple IIc and IIgs. The firmware for the Super Serial Card was written by Apple's Larry Kenyon. The IIc serial firmware was written primarily by Apple's Rich Williams. And the IIgs serial firmware was written primarily by Apple's Mike Askins.

Consistency with the Super Serial Card was not the highest priority in the design of the IIc or IIgs serial firmware. Particularly with the original Apple IIc, the limited amount of space for firmware was supposed to make emulation impossible (the IIc had to fit firmware for two serial ports and the mouse in same amount of address space the Super Serial Card used). Though he was supposed to design just a simple serial interface, Williams managed to squeeze much of the functionality of the Super Serial Card into the original IIc. He did such a good job that today people wonder why the clown who wrote the IIc serial firmware didn't emulate the Super Serial Card exactly, rather than wondering how the hero who wrote the IIc serial firmware managed to squeeze all that stuff into such a small space.

Modes and defaults. All three serial interfaces can operate in either a "printer mode" or a "communications mode." Printer mode is used for one-way transmission, as from your computer to a printer. Communications mode is used for two-way transmission, as from your computer to a modem and back.

The mode that an interface operates in has to be set up ahead of time—it cannot be changed with a command. With the Super Serial Card, you specify the mode by setting a dip-switch on the card itself. The manual tells you how. With the IIc, you specify the mode using the IIc System Utilities program. With the IIgs, you specify the mode using the control panel.

The IIc and IIgs default to printer mode on port 1 and communications mode on port 2. This is what you'll always get when you turn on the IIc. The IIgs, on the other hand, stores its control panel settings in battery RAM, so if you change these default modes it will remember your new choice while turned off. If you change the IIc power-on defaults with the System Utilities program, what actually happens is that some values in the "auxiliary memory screen holes" are modified. The new values will be remembered as long as the IIc is turned on. Every time you initialize a IIc serial port, the firmware looks in these screen holes to see what the current "dip-switch settings" are. You can modify these values with your own programs—for more information, see our November 1985 issue, pages 1.86 and 1.87.

In addition to mode, all three interfaces support a number of other variable characteristics. Each device figures out the default characteristics you want the same way it figures out its mode—dip-switch settings for the Super Serial Card, auxmem screen hole values for the IIc, and control panel settings for the IIgs. Some of these variable characteristics include baud rate, data format, parity, echo, line width, and handling of carriage returns and line feeds.

In most situations, your goal as a user should be to set these defaults to values that support your equipment and leave them alone. If you succeed at this, you don't need to know anything more about serial interfaces.

Command syntax. The rest of us will now learn about how to send commands that cause an interface to change its characteristics from their default values to something else.

The Super Serial Card and the serial ports on the IIc and IIgs can all be controlled by embedding "command strings" in your printed output. The idea is that the serial firmware will recognize and intercept these strings of ASCII characters, then act on them rather than passing them on to the printer or modem. This is a common method for controlling devices attached to a computer—your printer and modem probably have their own command strings that they react to. Even the Apple's disk operating system, when used from Applesoft, is controlled by command strings. Both the Basic Interface and the Advanced Interface (also known as Pascal 1.1—see last month's article for more information) on Apple's serial firmware respond to command strings.

Though common, the technique isn't foolproof. "Control-D(efeated)" in our December 1986 issue, page 2.85-86, discusses a number of specific problems with the embedded command string technique. On the other hand, the technique is quite flexible and allows for a great deal of compatibility. AppleWorks, for example, provides a way for you to enter an interface card command string that will be sent each time you print something. Since you are able to completely control the contents of the command string, you can adapt AppleWorks to any interface card that reacts to them. The alternative methods of controlling devices, such as POKEing controlling values into memory or CALLing command routines, provide finer control but less ease of use and less compatibility. The IIgs serial firmware, incidentally, does support control calls as well as command strings, as discussed here last month, but the Super Serial Card and the IIc allow only command strings.

Apple has used three different formats for serial interface command strings. All begin with a command character (usually control-I or control-A, more about this later). In the first format, the command character is followed by a single, non-control character, for example "control-I R." In the second format, the command character is followed by a decimal number, which is immediately followed (no space character allowed) by a single, non-control character, for example "control-I 80N." In the third format, the command character is followed by single, non-control character, a space, and either "E" (enable) or "D" (disable), for example, "control-I L E" (the IIc technical reference manual says there should be "no intervening space" with this syntax, however, the IIc actually ignores spaces; the Super Serial Card requires them).

The Super Serial Card, the IIc, and the IIgs each has its own special rules about whether extra characters, spaces, or lower-case are allowed within a command string. To keep your software as compatible as possible, always use upper-case characters and never include spaces or extra characters in a command string, except for the space that is required immediately before "E" or "D." (If you are writing *firmware*, on the other hand, you'll get the greatest compatibility by allowing the lower-case versions of command characters as well as upper and by making spaces optional anywhere in the command string.) Apple's firmware will accept commands whether they are sent in high-value ASCII or low-value ASCII. If you are writing firmware, make sure yours does as well.

An important difference between the Super Serial Card and the IIc and IIgs serial firmware is that the Super Serial Card *requires* a Return at the end of a command string. The Super Serial Card considers everything between the command character and the Return character to be part of the command string. None of this, *including the Return itself*, is sent to the printer or modem. Even if the command is unrecognized, the Super Serial Card will eat it all. On the IIc and IIgs, on the other hand, at least some portion of an unrecognized command is usually passed on to the printer or modem.

In addition, the IIc, IIgs, and most other Apple II interfaces do not require a Return at the end of the command string (the Super Serial Card is the non-standard device). If a Return is included, it is always passed on to the printer. Thus, any program that uses interface card commands and that is interested in controlling the position of the paper in the printer must figure out if it is using a Super Serial Card or not. If so, it should put a Return at the end of all commands (whoops—except three "parallel card" commands and the "change command-character command," all noted later). If the program is

running on a IIc or IIgs, on the other hand, Returns at the end of commands should always be suppressed (whoops—except after a special version of the line width command, more later).

For example, if you are setting up an interface card command string within AppleWorks, you have to know whether the command requires a Return or not. If it does and you leave the Return off, the command will either not be executed or will cause other problems. If it doesn't and you put one there anyhow, your paper will be advanced one line without AppleWorks knowing about it.

Another problem is that since the IIc and IIgs don't require a Return, they allow command strings to be strung together, such as "control-I 80N control-I L E." This doesn't work with the Super Serial Card because the required Return at the end of each command is missing.

The command character. As mentioned, each command string must begin with a "command character." The command character defaults to control-I in printer mode and control-A in communications mode with all three of Apple's devices. You cannot specify a different *default* command character. However, all three devices allow you to change the command character after initialization by printing the current character and following it with any other control character except a carriage return. "Control-I control-Z" (CHR$(9);CHR$(26)) for example, will change the printer mode command character to control-Z. "Control-I Return" (CHR$(9);CHR$(13)) does nothing. "Control-I control-Z Return" (CHR$(9);CHR$(26);CHR$(13)) changes the command character to control-Z and passes a Return to the printer—even with the Super Serial Card. On the IIc, "control-I control-@" (CHR$(9);CHR$(0)) has the undocumented feature of executing the primary feature of the "Zap" command, about which we'll talk more later. So don't use control-@. Another character that causes problems is control-D. It interferes with DOS. Various Apple manuals also recommend against control-A, -B, -C, -H, -I, -J, -L, -M, and -Y. Control-Z is frequently used.

The firmware in the IIc and in the IIgs resets the command character to its default value every time the firmware is initialized. This means that after every PR# (IN#) or INIT call (the INIT call is available under the Advanced Interface) the IIc/IIgs command character is reset to control-I or control-A. The Super Serial Card, on the other hand, *does not* reset the command character to the default value. With the Super Serial Card in printer mode, do a PR#1, change the command character to control-Z, and do another PR#1—the command character will still be control-Z. On the IIc and IIgs, your second PR#1 will reinitialize the command character to control-I. In this respect the IIc and IIgs are like most other Apple II interface cards—the Super Serial Card is the non-standard device.

This difference is at the root of a fair proportion of "printer" problems and is important to understand. It applies not only to the command character, but to *all* Super Serial Card default characteristics except "Zap" status. After a PR#1 or the equivalent, the Super Serial Card will reset itself to its dip-switch defaults *only if* this is the first initialization following power-up, following control-reset, or following the Super Serial Card's R(eset) command. In all other cases, PR#1 leaves the Super Serial Card's characteristics as they were when the firmware was last used.

Consequently, any program that uses the Super Serial Card and that changes the default settings must also *unchange* them. If it doesn't, subsequent programs, or even the same program run a second time, may find the Super Serial Card unresponsive to commands. For example, if a program changes the command character to control-Z and neglects to change it back to control-I, control-I commands from programs run later will not be recognized by the Super Serial Card.

If the same program is run a second time, for instance, it will send the control-I control-Z code again. The control-I will be sent to the printer and the control-Z will be recognized as the beginning of a command. Pretend the next thing sent is a command to add linefeeds to carriage returns, "control-Z L E." When the Super Serial Card sees the two control-Zs in a row it will send the second one to the printer. Thus, the control-I, the second control-Z, and the "L E" will be received by the printer—not the program's intention at all.

None of this is a problem with the IIc or IIgs, but to make your program compatible with the Super Serial Card and third party cards that emulate it, you must always change the command character back to control-I if you change it to anything else. Make sure that it gets changed back even if your user stops printing in mid-page.

If, as a user, you find a commercial program in your library that doesn't return the Super Serial Card to its default settings, the only thing you can do to set things right is to turn the computer off or press control-reset after running the program. There's a chance you could use the Super Serial Card's "control-I R" reset command (more later) but if the command character has been changed, even this command will go unrecognized.

The feature whereby two command characters in a row sends one of them to the printer works only on the Super Serial Card. Neither the IIc nor the IIgs support this, so don't use it.

While you might expect that all third-party cards would use control-I and control-A as the default command characters, it isn't so. The popular Grappler-Plus uses control-Y for commands sent to its Advanced Interface (see "Go, Logo, Go," September 1987, page 3.64 for a good example of the kind of problems this causes). Firmware authors—please stick with control-I and control-A.

Reset. All three interfaces under discussion here support the "reset firmware" command. This command tells the firmware to reset itself to its "dip-switch" (auxmem, control-panel) defaults the next time it is initialized. However, on the IIc and IIgs this always happens automatically, whether you've used the R command or not. The R command also disconnects the interface firmware. With a Super Serial Card, this will jolt your screen out of 80-column mode if it's in it. Thus, Reset is an ill-behaved command that has no useful effect except with the Super Serial Card. Nonetheless, any program that uses other command strings should issue this command as part of its QUIT routine, so that the next program that runs can assume a "normal" serial interface. The syntax is, for example, "control-A R." To avoid the 80-column jolt, try a program-ending sequence like this one:

```
9990 REM ...end of program
9991 HOME .                 : REM clear screen to avoid jolt on 80/40 switch
9993 PRINT CHR$(21)         : REM turn off 80 columns just to be safe
9995 PRINT CHR$(4);"PR#1"   : REM turn on serial port
9996 PRINT CHR$(9);"R"      : REM reset defaults and turn serial port off
9999 PRINT CHR$(4);"BYE"    : REM back to the program selector
```

Zap. The Zap command was designed to solve some of the problems that occur with devices that use embedded command strings. After you issue a Zap command, the interface firmware will stop intercepting command strings and will pass everything it receives on to the printer or modem. The Zap command is handy, for example, when you want to send strings of binary data to a printer for a graphics printout or for a downloaded character set. Without Zap, problems occur if your binary data accidentally includes a value that duplicates the interface firmware command code. That byte and an indeterminate number of the following bytes will get swallowed by the firmware and your picture or character set will have a hole in it. The syntax for Zap is, for example "control-I Z."

To turn Zap off, simply do another PR# (IN#) command or make the Advanced Firmware INIT call. This works even with the Super Serial Card. There is no command string for turning Zap off, obviously, because the firmware is ignoring all command strings when Zap is on.

Because the Zap command is typically used before sending binary data, many programmers make the assumption that in addition to telling the firmware not to eat any characters, it also tells the firmware not to burp any additional characters into the character stream. This is not the case.

Apple's serial firmware can be set up to automatically burp both linefeeds and carriage returns into a character stream. Linefeeds, if burped, are burped after Returns. Returns, if burped, are burped when a specified number of characters have passed without a Return. This is for line-length control. With the Super Serial Card, the Zap command has no effect on either linefeed or Return insertion. With the IIc and IIgs firmware, the Zap command automatically turns off Return insertion, but has no effect on linefeed insertion.

Thus, if you are using the Zap command to pass binary data to a printer or modem and you don't want occasional extra bytes inserted into your data, you should turn off both Return and linefeed insertion before executing the Zap command. If you do this with a Super Serial Card, don't forget to have your program turn the printer back on and issue the Reset command during its QUIT routine.

Linefeeds. A "linefeed" is a control character that tells a printer to advance the paper one line. Many printers require a linefeed after each carriage return, since all they do in response to the Return is move to the left edge of the paper. All three of Apple's serial interfaces have the ability to either add or not add a linefeed after each carriage return you send.

In general, the default setting for linefeed insertion is ON with printer mode and OFF with communications mode. However, the Super Serial Card allows you to control this default with a dip-switch in printer mode, and the IIc and IIgs, using the IIc's auxmem screen hole values and the IIgs's control-panel, allow default control in both printer and communications modes.

If the default setting isn't what you want or if you aren't sure of the default, you can force linefeed insertion to OFF with the K command. This is one of the three "parallel" commands that don't require a Return on the Super Serial Card. (If you put a Return after this command it will be passed to the printer.) On the Super Serial Card this command, like all parallel card commands, works only in printer mode. On the IIc and IIgs it works in communications mode, too. See "One of the first interface cards designed for the Apple II" in last month's issue, page 3.65, for more information on the heritage of this and the other parallel card commands.

Unfortunately, there is no universal way to turn linefeeds ON with Apple's firmware. The Super Serial Card, the 3.5 ROM IIc, and the IIgs support an enable/disable linefeed command that goes, for example, "control-I L E" (command-code, Linefeed, Enable) or "control-I L D" (command-code, Linefeed, Disable). The original IIc doesn't support this syntax. The original IIc firmware does support an L command, however, that will turn linefeed insertion on. It goes, for example, "control-A L." The 3.5 ROM IIc firmware sort of supports the original IIc's L command, but not very well (the 3.5 version seems to require either another command character or a Return after the "L;" if it gets a Return it eats it, although the original IIc firmware would pass it through). Neither the Super Serial Card nor the IIgs support the "L" version at all.

In addition to inserting linefeeds into the data it sends out, all Apple serial firmware can also delete linefeeds from data coming in from outside the computer. Obviously, this has no meaning in printer mode, so it applies only to communications mode. The default setting for this feature is a mish-mash. The Basic Interface on the Super Serial Card and both versions of the IIc firmware default to deleting incoming linefeeds. The Advanced Interface on the Super Serial Card and both interfaces on the IIgs firmware default to allowing incoming linefeeds to pass through unmolested.

With the Super Serial Card, the 3.5 ROM IIc, and the IIgs, you can use a "Mask linefeeds" command to control linefeed deletion. For example, "control-A M E" (command code, Mask, Enable) causes incoming linefeeds to be deleted; "control-A M D" (command-code, Mask, Disable) causes incoming linefeeds to be passed. With the original IIc, there is no way to change the default—incoming linefeeds are always deleted.

Screen echo, line width, and Return. "Screen echo" has to do with whether the data you send out from your computer appears on your screen as you send it. In the context of serial ports, this characteristic is also sometimes called "video." Some commands for turning screen echo on and off also affect "line width." Line width affects how often Returns are burped into the character stream being output. However, on some interfaces, line width isn't the only variable that controls Return insertion.

First let's look at Return insertion. The Super Serial Card, the IIc in communications mode, and the IIgs all default to NOT adding Returns when the specified line width has been reached. The original IIc firmware in printer mode (and most parallel card firmware), on the other hand, does insert a Return when the specified line width has been reached. This can be a problem when using condensed characters with software such as AppleWorks. At 17 characters-per-inch you can get 136 characters on an 8 inch piece of paper. If the interface firmware adds a Return after every 80 characters, however, what you'll see on your printout won't be what you expected.

In most applications today, you want Return insertion turned OFF. However, neither the Super Serial Card nor the original IIc have a command for doing this. With the Super Serial Card this isn't a major problem, since it defaults to no Return insertion, anyhow. With the original IIc, however, which defaults the other way, it is a problem. The IIc manual mentions a POKE you can use to turn off Return insertion, but we're not allowing POKEs in this article. The only command-based alternative for turning off Return insertion on the original IIc is to use the Zap command. There are *no* command-based alternatives for turning off Return insertion on the Super Serial Card, not even Zap does this—so don't turn it on unless you really need it.

While most applications work best with Return insertion OFF, LISTing Applesoft programs on a printer works best with Return insertion ON. This way the ends of lines that are more than 80 characters long are "wrapped" to the following line rather than being printed on top of the beginning of the line. The IIc and most parallel cards do this automatically. The Super Serial Card and the IIgs, however, require two commands to implement this. One is to set the line length to something other than zero and the second is to turn on Return insertion.

To turn ON Return insertion with the Super Serial Card, the command is "C," as in "control-I C." This syntax works only on the Super Serial Card, however. The 3.5 ROM IIc and the IIgs support an enable/disable command for this, as in "control-I C E." On both versions of the IIc you can also turn on Return insertion simply by setting the line width to a non-zero value. (The 3.5 ROM IIc's "C E/D" command sets the line width to zero when you choose disable and sets line width to the initialization default when you choose enable. Thus, if the default line width is zero, as it is in communications mode, the "C E" command has no effect. Not to worry, use one of the line width commands to set a positive line width instead.)

To summarize: with the IIc (and most parallel cards), Return insertion always occurs unless the line width is set to zero (unfortunately, however, the original IIc has no command for setting the line width to zero other than Zap). With all other Apple interfaces, Return insertion requires both a non-zero line width and enabled Return insertion.

Line width is controlled by a variety of commands, only one of which is nearly universal. The "I" command, as in "control-I I," sets the line width to 40 on the Super Serial Card (it also affects echo; more about that in a moment). On the IIc and the IIgs the "I" command has no effect on line width. Both versions of the IIc and the IIgs support a line width command that consists of a decimal number, between 1 and 255 inclusive, followed immediately by a Return. The Return is eaten by the firmware and the line width is set to the decimal number. This command has no affect on Echo. The Super Serial Card doesn't support this command, however.

The nearly universal version of the line width command is "nN," as in "control-I nN," where the small "n" is again a decimal number between 1 (IIc/IIgs) or 40 (SSC) and 255 inclusive. What prevents this command from being universal is that it doesn't work in communications mode on the Super Serial Card. Another feature of the "nN" command (which is one of the "parallel card commands" that never requires a Return after it) is that it disables Echo.

There are also a variety of ways to control Echo. The simplest is the Echo enable/disable command, as in "control-A E D," for example. This command isn't available on *any* version of the IIc, however, nor is it available from printer mode on the Super Serial Card. The "I" command mentioned earlier (which is the third and final of the parallel commands that never require a Return) enables Echo on all interfaces except the Super Serial Card in communications mode. In printer mode on the Super Serial Card, it also changes the line width. The "nN" command disables Echo on all interfaces except the Super Serial Card in communications mode. It also changes the line width.

When you are using a program such as AppleWorks, you want your interface card's Echo disabled and Return insertion off. (Echo disabled so that what you print doesn't appear on the AppleWorks screen; Return insertion off so that AppleWorks has complete control over Return placement.) The "nN" command is the universal "no Echo" command. Unfortunately, there is no universal "no Return" command—in fact, as we have seen, the Super Serial Card and the original IIc don't have any command for this at all. On some third-party parallel cards, "control-I 0N" does the trick. However, the Super Serial Card ignores this command completely; the IIc and IIgs respond by disabling Echo but leaving the line width unchanged. The closest thing to a universal interface card command for AppleWorks and similar programs would be "control-I 255N." This disables Echo and, while it doesn't turn off Return insertion totally, it will suffice until we have printers that can fit more than 255 characters on one line.

While AppleWorks and similar programs prefer Echo and Return insertion OFF, Applesoft programmers often prefer to have them both ON. This is so that you can see what you're typing after you enter a PR#1 command from the keyboard (if Echo is disabled what you type appears only on the printer), and so that long lines wrap correctly, as mentioned earlier. There are no universal commands to accomplish this, either. On the Super Serial Card you must execute the "control-I C" command to get return insertion to work at all. If you next do a "control-I I" to enable Echo, that command also sets the line width to 40. If you try to change the line width to something longer with a "nN" command, that will turn Echo back off. Gotcha—the Super Serial Card won't let you have both Echo and Return insertion unless the line width is 40. On the IIc, Return insertion works whenever the line width is non-zero, so an "nN" command to set the line width you want followed by an "I" command to enable Echo will get you where you want to go. Finally, on the IIgs, you first must enable Return insertion with a "control-I C E" command, set the line width you want with an "nN" command, and enable Echo with either an "I" or an "E E" command.

In communications mode, whether you want Echo or not depends on whether you are working with a "full duplex" or "half-duplex" host computer. Under the typical full-duplex arrangement, the host computer echos the

characters it receives from you back to you and it is the echoed characters that appear on your screen. (You type "A," it goes out your serial port, through your modem, down the telephone line, through the receiver's modem and serial port, to the host computer, which echos it back through its serial port and modem, over the telephone line, through your modem and serial port, and onto your screen—makes you appreciate the speed of electrons. Incidentally, if both computers are set up to echo everything they receive, the "A" flies back and forth like a ping-pong ball and the communications link seems to freeze up.) Under full-duplex you want your local Echo disabled, or else all the characters will appear on your screen twice. A half-duplex host, on the other hand, does not echo your characters back to you. For this you want local Echo enabled, or else you won't be able to see what you are typing. Note that in communications mode the Super Serial Card does support the "E E/D" command, as does the IIgs. The IIc and the Super Serial Card in printer mode don't have this command; again, your only recourse is to control Echo with the "I" and "nN" commands.

Baud, data format, and parity. For most applications there is no reason to have to change baud, data format, or parity by command. You should be able to set up your system so that the correct characteristics are chosen by default. With the Super Serial Card, you choose the defaults by setting dip-switches, with the IIgs you choose the defaults with the control panel. With the IIc you choose the defaults with the System Utilities program. If you'd rather not have to run the System Utilities program every time you turn your IIc on, simply make your printer, modem or whatever match the IIc's default defaults—9600 baud, 8 data bits, no parity for port 1; 300 baud, 8 data bits, no parity for port 2 (for more information on data format and parity see Uncle DOS's introductory comments in last month's issue, page 3.69).

If you must change one or more of these characteristics by command, at least you'll find near perfect consistency in the commands available to do this. Baud is changed with an "nB" command; data format with an "nD" command, and parity with an "nP" command, where "n" is:

nB=baud rate	nD=data format data/stop bits	nP=parity
0* use default	0 8 1	0 none
1 50	1 7 1	1 odd
2 75	2 6 1	2 none
3 110	3 5 1	3 even
4 134.5	4* 8 2 or 1	4* none
5 150	5 7 2	5* MARK
6 300	6 6 2	6* none
7 600	7* 5 2 or 1-1/2	7* SPACE
8 1200		
9 1800	* NOTES: 0B resets baud to dip-switch/control-panel	
10 2400	default on SSC/IIgs, drives IIc crazy.	
11 3600	4D has 1 stop bit with parity options 4-7.	
12 4800	7D has 1-1/2 stop bits with parity options 0-3.	
13 7200	SSC and IIc only	
14 9600	4P-7P not available on IIgs	
15 19200		

As almost always, the Super Serial Card expects a Return after these commands, the IIc and IIgs don't.

Handshaking. Handshaking is a morass I'd just as soon not get into, but since serial devices don't work well without it, I guess we'll have to. Here's the idea of handshaking—you are printing an AppleWorks spreadsheet on your printer and your interface and printer have been set up to communicate at 9600 baud. Although your printer can *communicate* at that speed, it can't actually *print* anywhere near that fast. Most printers have at least a small amount of memory for storing a few of the characters waiting to be printed, but for large amounts of characters the printer has to have some way to tell the computer to stop and wait. Once the printer has caught up, it also needs a way to tell the computer to proceed. This is what handshaking is all about.

Serial devices have several different hardware protocols that go by names such as "data carrier detect" (DCD), "data set ready/data terminal ready" (DSR/DTR), and "request to send/clear to send" (RTS/CTS). These protocols involve electrical signals on special wires running between the sending and receiving devices. With the Super Serial Card and the Apple IIc the commands for changing hardware protocols involve a pair of wire cutters and a soldering iron. The Apple IIgs allows you to independently turn DCD and DSR/DTR handshaking on and off using the control-panel defaults or using control calls to the Advanced Interface, but provides no command strings for this.

In addition to *hardware* protocols, there are *software* handshaking protocols. These involve the receiving device sending command strings to the sending device that tell it when to start and stop. The version of this protocol used by Apple's interfaces is called XON/XOFF. It is available on all the interfaces we're discussing here except the original IIc, which doesn't support it. The command for turning it on and off is "X E/D." It always defaults to off.

Applesoft TABs. The Super Serial Card and the IIgs include a command that causes the Monitor's horizontal position counter to be left equal to the column count. Normally both the Super Serial Card and the Apple II 80-column firmware force this counter to zero. To get the Applesoft PRINT TAB command to work correctly when sending stuff to your printer, it is necessary to PRINT CHR$(21) to the screen to turn off the 80-column firmware and then to issue the serial interface Tab command. This command has the enable/disable syntax. With the Super Serial Card the command letter is "T," with the IIgs it is "A," so for example, "control-I A E" would enable Tabs on the IIgs. It is also a good idea to disable Echo when using the command, otherwise the first few lines of your Applesoft program, which lie adjacent to the memory area used for the text screen, can get poked full of holes as characters are displayed beyond the 40th column of the 40-column screen.

Neither version of the IIc has a Tab command. For some more universal solutions to this problem, see "PRINT TAB alternatives" in our March 1987 issue, page 3.14.

Communications mode. All three interfaces include "dumb terminal" software. This lets you talk over a modem quite easily. None of your session is saved anywhere, however, so you can't use this for "downloading" material any faster than you can read it or copy it onto a sheet of paper. For all three interfaces, the command for entering the terminal is "T," as in "control-A T." To exit the terminal, use "control-A Q."

All three interfaces also support "remote-control" terminal commands. This means that if a control-T comes in from a remote device, your serial firmware will jump into terminal mode and stay there until it sees a control-R. I can't quite figure out what the practical value of this feature is, but somebody somewhere obviously once thought it was important. It can be a troublesome feature if you are trying to receive binary data through the serial firmware and the data accidentally includes a control-T, because further data (up to the next control-R) will be lost. A similar problem occurs if you have XON/XOFF handshaking enabled and you receive a control-S or control-Q (the XOFF and XON characters). In that case, at the very least the control-S or control-Q will be eaten. In some cases, all data after the control-S and before the next control-Q may be lost.

With the Super Serial Card you can avoid both problems by turning off XON/XOFF protocol (conversely, this means control-T works only if XON/XOFF is enabled). On the IIc, a Zap command will prevent control-T and control-R from working. The IIgs doesn't have a way to prevent control-T and control-R from working that I can find.

Another potential problem that occurs when you are receiving data from a remote device is corruption of the data by accidental taps on your keyboard. Normally the serial firmware accepts input from the remote device and the keyboard simultaneously. A program can temporarily disable the keyboard by issuing a "control-A F D" command. After this command, keyboard pecks will be ignored. The keyboard can be re-enabled with "control-A F E," but obviously this will have to come from your program—you can't type it in and expect anything to happen.

One other feature you may find useful in communications mode is the power to send a BREAK signal. This signal consists of 233 milliseconds worth of zero bits and is used by some computer systems for signoff. On the Super Serial Card, the command for this is "control-A B Return." On the IIc and IIgs it's "control-A S." I can tell from the Super Serial Card and IIc source code that their versions of BREAK are shorter than 233 milliseconds when an accelerator is used; I suspect this was fixed on the IIgs but I can't figure out a way to test it.

Other commands. The Super Serial Card supports several more commands not already mentioned. Three of these are for adding delays after carriage returns, linefeeds, and formfeeds. Printers without memory buffers require these delays or they skip characters that come in while the printer is moving the print head or the paper. Neither the IIc nor the IIgs support these commands. The Super Serial Card also has a command that allows your Apple to be connected to a modem in one slot and an external terminal in another and a command that can cause conversions between upper- and lower-case.

Of more interest is a command new the IIgs, a command that enables input and output buffering. Issue a command such as "control-A B E" and you suddenly have 2,048-byte buffers for both input and output (assuming the memory manager has that much memory available—if not, you'll get

128-byte buffers). The size of these buffers can be changed using Advanced Interface control calls.

Here's how the buffers work—whenever an incoming character arrives or an outgoing character has been transmitted, the serial chip will send an interrupt signal to the firmware. The firmware collects incoming characters and puts them in the input buffer and takes characters out of the output buffer and transmits them.

Like other IIgs serial firmware features, buffering status after port initialization can be controlled with the control-panel. Note, however, that if your control-panel default is set to no buffering and you set up buffers by command, those buffers will disappear if you reinitialize the port with a PR# command or the equivalent.

Applesoft programs can get incoming characters from the input buffer or put outgoing characters into the output buffer with normal INPUT and PRINT commands. The advantage of an input buffer is that normally Applesoft can't do INPUTs fast enough at high baud rates to capture all incoming characters. But with a buffer and handshaking, the serial firmware effectively slows down the rate of transmission so that Applesoft can handle it error-free. The advantage of an output buffer is that you can PRINT a chunk of text and go about your business while the firmware sends it out at a slow baud rate.

Using Advanced Interface control calls with the IIgs serial firmware, you can build on the buffering routines to do fancy stuff such as create large printer buffers for background printing. The IIgs can feed characters to your printer at whatever speed it requires while you go on about your business.

If last month's article made you sleepy, this month's might give you insomnia. I hope not. Pull up the covers, Bobo.

Ask (or tell) Uncle DOS

If you've ever daydreamed of having a job like Uncle DOS, if you're good, and if you're committed to the Apple II, send a resume to Jim Merritt, Manager; Apple II Developer Technical Support MS-27-T; Apple Computer Inc.; 20525 Mariani Ave; Cupertino, CA 95014.

Mopping up EMI

Television interference (TVI), radio interference (RFI), and broadcast interference (BCI) are three of the biggest problems that plague home computer owners, as mentioned in last month's letter "Apple Makes Family Grumpy" (page 3.70). As a group, they are known as electro magnetic interference (EMI). I am a ham radio operator and have had to learn quite a bit about what causes the problems and what can be done to fix them. Sometimes, fixes are relatively easy. Other times they are simply impossible.

Some reading material that ***Open-Apple*** readers who have EMI problems might find interesting are the *Apple II-Plus/IIe Troubleshooting & Repair Guide*, by Robert C. Brenner, pages 189-194 (H.W. Sams, $19.95), *Radio Frequency Interference* (American Radio Relay League, $4), and the October 1987 issue of *PC Resource* magazine.

Computers create EMI because they have oscillators—components that create electronic signals that "vibrate" on and off very rapidly. The Apple IIe's main oscillator vibrates at a speed of just over 1 MHz (1 million vibrations per second). The Apple II's power supply also generates electronic vibrations in the 10-100 kHz range (10 to 100 thousand vibrations per second). If you have a speed-up card, it also has an oscillator.

In a perfect world, the frequencies mentioned would not be in a range that would cause interference with radios and TVs. However, the oscillators also create harmonics that will extend far beyond the fundamental frequency. These harmonics usually are where EMI and other problems occur.

EMI can be "conducted" (passed through wires—usually the AC power line) or "radiated" (passed through space like radio signals). Some simple ideas that can solve conducted EMI are to make sure the computer and TV are on separate AC circuits and to connect traps or line filters between the power outlet and the TV or computer or both.

If your problem is radiated EMI, some simple solutions may be to locate the computer farther away from the TV, to rotate or reposition your TV antenna, to replace the TV's twin-lead antenna cable with 75 ohm TV cable, or to subscribe to cable television.

Or you can unplug all the cables from your computer and see if that helps. Cables can and do act like antennas—they radiate the computer's internal EMI to the outside world. With the computer on and no cables connected, see if you still have interference. If not, you can reduce radiation from your cables (including the power cord) with "ferrite cores." If your computer dealer has a sharp technician, they may keep them in stock. A good electronics supply house should have them too. Computer Radio, Box 282, Pine Brook, NJ 07058 sells a split-ring choke that is very easy to install and is specially designed for computer EMI applications. For $17 ppd you get a package of four cores. These cores are split so that cables with connectors can be wound in them. Wind the cable two to four times around the core.

From this point on fixes get more difficult.

My Apple IIe is by far not the worst computer in my house. I have an MS-DOS clone that completely obliterates all TVs in the house. But that's another story.

J. Craig Clark Jr.
Ham Radio Magazine
Greenville, N.H.

Patch instructions patchy

...I can't get the "AppleWorks as copy machine" patch (October 1987, page 3.71) to work. The patch didn't include loading directions for the SEG.M1 file...

Lee Hayward
Denver, Colo.

...October's "AppleWorks as copy machine" letter instantly appealed to me. But there must be a few introductory steps before one types the patch instructions. This poor soul isn't even able to load SEG.M1...

Robert J. Netro
Canton, Ohio

The secret to the patch as published is that you don't have to load SEG.M1. I should have made that clear. Since the patch is only one byte long, Bird just pokes that new value, 255, into memory at byte 768, then "BSAVEs" that byte into the correction locations in the program. Here's an explanation:

```
cmd    filename   filetype  length   adr     byte
BSAVE  SEG.M1,    T$00,     L1,      A768,   B36074
```

Notice that the length of what's being saved is 1 byte long. The A parameter tells Basic.system to look for that byte at memory cell 768. The B parameter tells Basic.system to stuff that byte into the 36,074th byte of SEG.M1, which has a file type of $00 (if you don't specify the file type, Basic.system will give you a FILE MISMATCH error, since BSAVE normally expects to work with a BINary file).

Under DOS 3.3 this command syntax would erase the old SEG.M1 and replace it with a 1-byte file. Basic.system, however, leaves SEG.M1 just as it is except for overlaying the 36,074th byte with the new value. For more on this see "The binary savior," and "It's painful to B misunderstood" in the July 1985 ***Open-Apple****, page 1.51.*

All AppleWorks patches should be performed on backup copies, of course, in case one of us makes a mistake. If you use Pinpoint or one of the AppleWorks expansion programs it is usually necessary to make this patch, and others like it published here, to an original copy of AppleWorks, then reinstall Pinpoint or your expansion program. The reason this is necessary is that Pinpoint and the expansion programs tend to move things around inside the AppleWorks files; what the patch expects to change at byte 36074 may now be elsewhere inside the file.

Two lines too cryptic

The two-telephone discussion in your July 1987 issue (page 3.42-43) is intriguing, but too cryptic for me. Could you explain it again?

Gary Smith
Claremont, Calif.

Call your local phone company and order a "second line" for your house. This line will be separate from your first and will have a different phone number. The phone company will run a new cable to your house that will end on an outside wall somewhere in a box called a "network connection." This is where the phone company's property ends and yours begins.

Somewhere inside your house you should be able to find a "junction" where a cable running from your first phone's network connection joins cables coming from all the phone jacks in your house. If you can't find this junction get professional help.

Run a new cable (available from Radio Shack, hardware stores, phone stores...) from your second line's network connection to the interior junction. At the network connection, attach the new cable's four wires according to the colors listed on the network connection box.

At your junction box, disconnect the yellow and black wires from your first telephone line and leave them unconnected. Connect a multimeter to the remaining yellow and black wires that run to your phones from the junction box. They should have no voltage and infinite resistance. If they don't, get professional help.

Now connect the red wire from your new phone line to the yellow wires running to your phones. Connect the green wire from your second line to your phones' black wires. Leave the new phone line's yellow and black wires unconnected.

Every jack in your house now has both phone lines ***available****. Existing phones will still be connected to your old number. To hook a phone to your new second line, you either need a two-line phone that will plug into a modular jack and receive both lines, or you will need a "two-line modular jack" to stick into an existing jack. It will give you two outlets to plug phones or modems into—one for the old line and one for the new line.*

Frigid flaw

Bill Basham's trick of wiping out a byte in the keyboard microprocessor's RAM to fool the Memory Manager into doing a "frigid" reboot is extremely dangerous ("Reboot, cold reboot, frigid reboot," July 1987, page 3.48). This byte is currently sitting at the bottom of the stack used by the keyboard micro. If Apple ever does a revision to the keyboard micro software, this location is almost guaranteed to change. Since the keyboard micro is tightly wound into the control panel and battery backed up RAM, Basham's trick could wreak havoc with future revisions and generations of the IIgs.

The reason the Memory Manager uses the keyboard microprocessor's RAM to determine whether the IIgs has been turned off is that the regular RAM in the IIgs has an uncanny ability to retain its state even after the machine has been turned off. When prototypes of the IIgs were powered off, then quickly turned back on, the Memory Manager performed its power-off check, which usually verified, and assumed that the RAMdisk should be preserved. The system would then boot from the RAMdisk, which would crash the system, since the RAM was not 100 per cent intact.

To fix this, I came up with the idea of using the RAM in the keyboard microprocessor, which is a static RAM that clears quickly when the power is turned off. But since the keyboard RAM test was added after the keyboard micro's code was done, it required the use of an unused byte in the stack area.

Peter Baum
Ninth Wave
Cupertino, Calif.

As an Apple employee, Baum designed the "Apple Desktop Bus," which is used to connect the keyboard, mouse, and other input devices to the Apple IIgs. He now runs an independent design and consulting firm and writes a monthly hardware column for ***Call - A.P.P.L.E.***

Open-Apple

[redacted]

is written, edited, published, and

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

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

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 single back issues are currently available for $2 each; bound, indexed editions of Volume 1 and Volume 2 are $14.95 each. Volumes end with the January issue; an index for the prior volume is included with the February issue. Please send all subscription-related correspondence to

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

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

ISSN 0885-4017 **Source Mail: TCF238**
Printed in the U.S.A. **CompuServe: 70120,202**

Expanding Applesoft

In response to Dr. Stephen White's questions in the October ***Open-Apple*** about Applesoft memory limitations, there is a suggestion I might make.

If all that is really required is additional memory for variables and strings, may I suggest the use of *The Beagle Compiler*? It's not ProDOS 16, but the compiled code takes up less space than the original program. And while the compiled code still has to fit in the "lower 48," variables can be stored in the alternate 64K memory bank, in extended memory in auxiliary-slot cards like RamWorks, or in standard-slot memory cards like RamFactor. This allows the use of *much* larger arrays of data.

I have used string arrays dimensioned to more than 2,000 strings with numerous other dimensioned arrays of similar size for floating point and integer variables. The program itself was on the order of 24K in length. This will go a long way to solving a lot of problems. Incidentally, even disk-to-disk compilation is fast (measured in seconds) and execution times are greatly increased by the compiler.

Robert L. Myers
Fayetteville, W.V.

BASIC for the IIgs

Contrary to your assertion on page 3.71 of the October issue that Apple has no plans to make BASIC available on the IIgs, the Apple Programmer's and Developer's Association was selling a beta version of *Apple IIgs BASIC* at AppleFest.

According to a product data sheet I picked up, the language is interpreted and supports structured programming (DO/WHILE, DO/UNTIL, labels rather than line numbers for GOTO and GOSUB); multi-line, named procedures and functions with parameter passing, local labels, and recursion support; easy redirection of input and output (character devices, such as printers, and block devices, such as disks, are both treated as "files"), PRINT USING and INPUT USING with IMAGE statements; support for single- and double-precision floating point arithmetic using Apple's SANE (Standard Apple Numerics Environment) tools, as well as 16-, 32-, and 64-bit integer variables; arrays of up to 32,767 elements per dimension and up to four megabytes in size; complete support for the IIgs Toolbox; and a licensable runtime version of the interpreter that allows distribution of applications to people who don't have their own copy of the IIgs BASIC software.

Some conversions may be necessary for Applesoft programs. Version 1.0 should be released to the public during the first quarter of 1988.

Cliff Tuel
San Jose, Calif.

What I meant last month was that an enhanced version of ***Applesoft*** *for the IIgs wasn't in Apple's plans, not that an enhanced version of* ***BASIC*** *wasn't to be. In retrospect, I should have made that clearer, as lots of people assume the two words mean the same thing.*

The first-peek beta version of ***Apple IIgs BASIC*** *is available to APDA members for $50 plus $3.19 shipping (Apple Programmer's and Developer's Association, 290 SW 43rd St, Renton, WA 98055 206-251-6548; membership is $20 a year U.S., $25 Canada/Mexico, $35 elsewhere). As you mention, the finished product is expected sometime next year. A compiled version of this language is also under development, but this will come from TML Systems in Jacksonville, Fla.*

At AppleFest, APDA also announced the availability of the first non-beta version of the ***Apple Programmer's Workshop*** *($100 plus $4.19). This provides what the professionals like to call a "command shell," which is fancy words for prompt-line-based file-manipulation commands such as those available at the Applesoft prompt (catalog, delete, rename, exec, PR#, and so on), as well as a bunch more Basic.system doesn't offer.* ***APW*** *also includes a full-screen text editor for writing programs, a 65816 macro assembler, and a "linker," which can make one large executable file from separate program files created by any combination of the assembler or other* ***APW*** *languages (Apple's* ***APW C****, currently in beta version 7 for $75 plus $1.31; and* ***TML Pascal****, $119.95 plus $1.50, are the only* ***APW*** *languages at the moment).*

Beta versions of ***APW*** *included a debugger and some desk accessories for peeking at data used by the memory manager and system loader during program execution. The debugger, which is still in its beta version, is no longer a part of* ***APW****, but is available separately under the name* ***Apple IIgs Debugger*** *($15.95 plus $.94).*

At AppleFest, APDA also released the ***Apple IIgs System Disk v3.1*** *($12.50 plus $.69), the Addison Wesley version of the* ***Technical Introduction to the Apple IIgs*** *($7.50 plus $1.50), and a first-peek photocopied version of the* ***Programmer's Introduction to the Apple IIgs*** *($37.50 plus $3.40).*

Previously available were the Addison-Wesley versions of the ***Apple IIgs Firmware Reference*** *($18.50 plus $2.50) and the* ***ProDOS 16 Reference Manual*** *($21.95 plus $2.75). Also available in a photocopied beta draft is the* ***Apple IIgs Toolbox Reference*** *($60.00 plus $8.00).*