Open-Apple

January 1988
Vol. 3, No. 12

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

Releasing the power to everyone.

To the stars through the Apple II

It was a hotel breakfast and my wife and I, as usual, each ordered a large orange juice, small for the kids. The waitress soon came back with huge 16-ounce glasses for my wife and I, and the kids got the 8-ounce glasses we expected for ourselves. "Toto," I mumbled, "we're not in California anymore."

And, in fact, we weren't. We were in the Sun Dome Best Western on the outskirts of Hutchinson, two hours east of Dodge City, an hour northwest of Wichita, or four, hard, two-kids-in-the-back-seat hours southwest of Overland Park. We were in Hutchinson to spend Thanksgiving at my sister's house.

Hutchinson is no Big Apple, of course, but neither is anyplace else outside of Manhattan borough. Manhattan Kansas calls itself the Little Apple, but Hutchinson, perhaps wary of the trademark police, doesn't claim to be any kind of apple whatsoever. In fact, if you don't count my sister's house, Hutchinson has just three major attractions.

First of all, Hutchinson is the world-wide headquarters for Dillons stores. Possibly only a few of you can appreciate the spiritual renewal a grocery store connoisseur feels on a pilgrimage to this Mecca. A visit to any Dillons, to browse the wide aisles lined with everything from apple pies to zwieback toast, always makes me feel close to heaven. In Hutchinson, that's the only kind of grocery store they have.

Secondly, every summer Hutchinson is the site of the Kansas State Fair. Things are kind of quiet over at the fairgrounds Thanksgiving weekend, however. No amazing snake women, no cowboys on horseback, and no road apples.

The third and greatest attraction of Hutchinson is the Kansas Cosmosphere and Space Center. This is one of the world's handful of major aerospace museums, jutting from the plains of Kansas just where you'd least expect to find it. The museum has over $100 million worth of space artifacts on display. For example, in the center of the main hall is a complete lunar lander. Outstanding exhibits trace the evolution of space flight from man's first liquid fuel rockets to actual spacecraft from the U.S. Mercury, Gemini, and Apollo programs.

The space capsules are displayed in such a way that you can get up close and see into the cockpits. Hands-on computerized exhibits let you experience such things as the sounds and window view during your choice of several lunar landings or ascents. A computerized Astronaut Data Bank answers questions and displays a digitized image of your favorite astronaut. A computerized display explains the functions of the controls in the Apollo command module. A computerized lunar geology exhibit displays your choice of videodisk photographs of the moon's surface.

During the summer, the space center runs seven one-week sessions of a summer camp called the Future Astronaut Training Program. Any student about to enter the 7th, 8th, or 9th grade can attend. At the camp, each student gets a chance to fly a variety of computerized trainers and simulators, including a full-size Manned Maneuvering Unit (jet pack), a flight trainer, and an 80 per cent scale Falcon Space Shuttle flight deck. Each of these simulators is big enough to sit in, and each is connected to computer and mechanical equipment that makes it raise, lower, tilt, and roll in response to student commands. The kids in the shuttle simulator are even directed by other kids in a computerized Mission Control simulator. (For more information about the center or the camp write to 1100 N. Plum, Hutchinson, KS 67501, or call 316-662-2305.)

But why do you suppose I'm doing a museum review here in *Open-Apple*? That's right,... *every computer in the museum is an Apple II.*

From the Omnimax wrap-around theater's reservation system, which is built into a counter in the lobby, to the accounting and administration offices, to the computers controlling the equipment that tilts and rolls the simulators, to the lunar geology videodisk display—computers are everywhere, and each one is an Apple II.

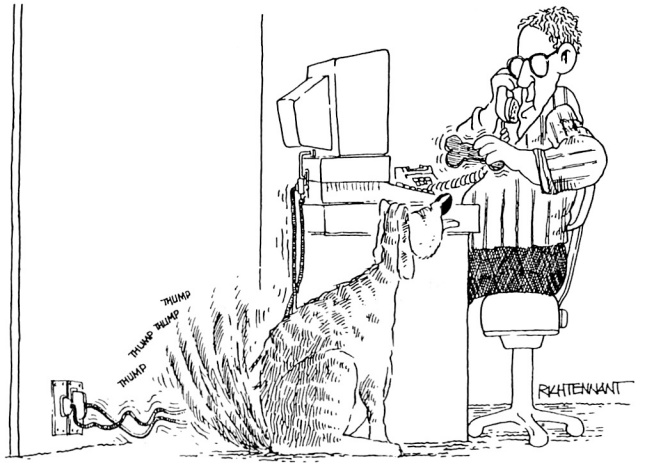
Many of the computers are in central equipment closets, to make servicing easy, and are hooked to the exhibits by long keyboard and video wires. Displays that have just a few buttons are actually connected to game paddle ports via resistor arrays so that when a button is pushed what the Apple II actually "sees" is a game paddle movement. According to the center's computer specialist, Ron Bloss, such an arrangement is immune to noise and allows the computers to be connected to the displays with hundreds of feet of cable.

I asked Bloss whether they were using any accelerators with their Apples. "No," he said, "we haven't seen a need for that yet, except maybe in accounting."

Even though the space center uses nothing but Apple IIs and is a high-technology education-based institution, Apple Computer doesn't appear on the long list of corporate supporters just inside the door. "We've tried to get some recognition from Apple," Bloss said, "but haven't gotten any response from them. They did finally let us purchase equipment through their education program, but other than that they've just ignored us."

And what about the future of the Apple II at the center? "As a matter of fact, we are about to decide whether to start using MS-DOS based machines," Bloss said. "When you can get an XT turbo clone with a hard disk for about half the price of a IIgs without a hard disk, it becomes very difficult to justify Apples to the board of directors. Besides, we need a network, and while we've heard rumors of AppleTalk on the Apple II, we can't make plans based on rumors."

No, Toto, this sure isn't Cupertino. You can't smell Apple's marketing smoke out here on the plains of Kansas and you can't see Apple's vision. But you can get a taste of what life's really like in the Apple II kingdom. You can shake hands with people who have built themselves a small Smithsonian-quality museum (and the largest tourist center in Kansas) on the back of the Apple II. And you can hear the crunching jaws of the clones, eating their way into Apple's markets.



"I TELL YA I'M STILL GETTING INTERFERENCE—
- COOKIE, RAGS? RAGS WANNA COOKIE? -
THERE IT GOES AGAIN."

Claris and its cash cow

Claris, Apple's infant software company, was featured in the November issue of *Outside Apple*, Apple's newsletter for third-party developers. Apple has seeded Claris with four Macintosh programs and AppleWorks. According to independent reports of retail software sales, AppleWorks by itself outsells the four Macintosh programs combined by a ratio of 3 or 4 to 1. You'd never know AppleWorks was so important to the company, however, based on its public statements.

For example, Kyle Mashima, Director of Product Development for Claris, notes in the *Outside Apple* article, "Adding one more significant publisher *is* a big deal. There aren't that many on the Macintosh side." Well, for starters there's Microsoft and Lotus, while on the Apple II side we have...well, we have...

Will Claris do anything at all in the Apple II market? "An Apple II product would have to be really exciting, have broad appeal, and be a menu-driven, integrated package for us to consider it," Mashima says, "But would we? Sure." Do you think Mashima has ever booted up AppleWorks? It's already an excellent example of a really exciting, broadly appealing, menu-driven, integrated package. Or by "menu-driven" does he mean "Xerox interface"? Do you suppose Claris will limit itself to integrated packages in the Macintosh market as well?

Apple itself has taken most of the profit out of the productivity portion of the Apple II software market for the last three years. In the process it has decimated Apple II productivity developers. Into this wasteland enters Claris with the intention of soaking the remaining profit out of its AppleWorks cash cow and wasting it trying to compete with insignificant Macintosh developers like Microsoft and Lotus. The only thing I don't understand is where Apple and Claris think new Apple II productivity software will come from.

Apple has won on all the chips on the Apple II software table and now wants to leave the game. This would make sense if Apple didn't also own the casino.

IIgs system disk 3.1 available

Apple dealers have had a new system disk for the IIgs for weeks already, and they'll let you copy it for free. However, the new disk, called system disk 3.1, still isn't shipping in the box with new machines. Dealers don't have the documentation that goes with this disk yet, either.

I haven't discussed the IIgs system disk in ***Open-Apple*** before, so let's go over a few basics. First of all, learn to distinguish between disks that have the volume name /SYSTEM.DISK, disks that have the potential to be system disks, and disks that really are System Disks.

Here in ***Open-Apple***, the "System Disk" (capitalized) is the disk you booted from. This could be a ProDOS 8 disk, such as the AppleWorks startup disk, or a ProDOS 16 disk, such as the new /SYSTEM.DISK your dealer will let you copy. (It could also be a DOS 3.3, Apple Pascal, CP/M, or MS-DOS disk.) If you boot the IIgs with a ProDOS 8 disk, you cannot run ProDOS 16 programs without rebooting. If you boot the IIgs with a ProDOS 16 disk (that has all the right files on it), you can run either ProDOS 8 or ProDOS 16 programs.

Important—ProDOS 16 works best when the System Disk (again, the disk you booted from) is "online" at all times. ProDOS 16 applications look on the System Disk for such things as fonts, desk accessories, RAM-based tools, and various kinds of drivers. If you take the System Disk out of the drive you'll frequently be prompted to reinsert it. Anyone who ever used an original, one drive, 128K Macintosh will tell you how frustrating this is.

Thus, it takes at least two drives to use ProDOS 16 programs efficiently—and if you have a IIgs with two 3.5 drives, one of them is pretty much always going to have your System Disk in it when you run ProDOS 16 programs. Now, 3.5 drives are nice *storage* devices, but they are relatively slow for this kind of use.

I recommend you buy a RAMdisk and just one 3.5 drive instead. Move all the needed /SYSTEM.DISK files over to the RAMdisk and boot from it. Your computer will operate much faster and you'll spend less money if you buy and install the RAM chips yourself. Now all you need is an easy way to load the RAMdisk in the morning. I'll get to that after we discuss exactly which system disk files you'll need to put onto your RAM-based System Disk.

(You have two options for a RAMdisk. One is to add lots of memory to the IIgs memory expansion slot and use a portion of it for a RAMdisk. To run ProDOS 16 programs well, however, you need a minimum of 512K of "desktop" memory; 1 megabyte is better. Any memory you use for a RAMdisk is *in addition* to this. Thus, you need 1.5 to 2 megabytes of gs memory if you want to use the built-in RAMdisk. In addition, if you want to be able to boot from this disk after booting from a 3.5 you have to fiddle with the control panel. A second option is to use a standard slot-based RAM card, such as a RamFactor. The problem with this is finding a slot to put it in. Slot 7 works well unless you need it for AppleTalk or a hard disk; choices after that become very difficult.)

(And speaking of a hard disk, that's another good place to put your System Disk. But the hard disk, obviously, must be your boot device for this to work.)

Here are the files that are in the root directory of the new system disk 3.1:

```
Filename          Blocks Type   Date

PRODOS                39  SYS  14-JUL-87  PQUIT, operating system loader
SYSTEM                 1  DIR  23-SEP-87  tools, fonts, and much more
SYS.UTILS              1  DIR   5-NOV-87  disk utilities
BASIC.SYSTEM          21  SYS  18-JUN-84  old faithful, V 1.1
BASIC.LAUNCHER         3  SYS  12-JUL-87  it takes longer to load than to run
APPLETALK              1  DIR  23-JUL-87  AppleTalk support software
ICONS                  1  DIR  23-JUL-87  put icon files in this subdirectory
DIALOG.ICONS           8  $CA   9-APR-87  what is this and why is it here?
COPY.ME                1  BAS  16-JUL-87  a do-nothing file used by tutorials
FINDER.DATA            1  $C9   5-NOV-87  the Finder created these to remember
FINDER.ROOT            1  $C9   5-NOV-87    desktop colors and positions
```

The PRODOS file on a IIgs system disk, as we've explained before, is quite different from the PRODOS file on a IIe or IIc system disk. The equivalent of the IIe/IIc PRODOS is actually in a file called P8 on the IIgs, which we'll see later in the SYSTEM subdirectory. On the IIgs, the PRODOS file is an "operating system loader." First it relocates a part of itself called PQUIT into an area of memory (the upper reaches of banks $E0 and $E1) where it will reside permanently. Next it loads ProDOS 16. Then it executes the stuff it finds in a subdirectory called SYSTEM.SETUP, which we'll find later in the SYSTEM subdirectory. Finally, it figures out what program it should run next and passes control to PQUIT. It's PQUIT, not PRODOS, that actually loads and runs the selected program and the operating system it requires.

The rules for "figuring out what program it should run" are these. First choice goes to a ProDOS 16 application program in the SYSTEM subdirectory called START. If there is no such file, the rules say to look in the main, or root, directory for a file that either is a ProDOS 8 system program (type $FF) with a filename that ends in .SYSTEM, or a ProDOS 16 application program (type $B3) with a filename that ends in .SYS16. Whichever is found first is selected. If no suitable files are found, the user gets a sliding apple screen and has to reboot with another disk. If you try to boot a ProDOS 16 system disk on a II-Plus, IIe, or IIc, you'll get a similar error message saying you can't do that.

Now, back to the /SYSTEM.DISK root directory. The second file shown is the SYSTEM subdirectory. We'll look inside it in a moment. It holds a number of disk-based program and data files that are needed to run ProDOS 16 applications.

The SYS.UTILS subdirectory holds the newest version (3.0) of Apple's 80-column, 64K, ProDOS 8 system utility program. This is the program you use if you want to convert files between ProDOS, DOS 3.3, DOS 3.2, or Apple Pascal. (It will also catalog CP/M disks, but cannot convert CP/M files.) The file conversion process is quite easy—just use the normal COPY FILES option to copy a data file from a disk of one operating system to a disk of another (program files, of course, will be converted only in a "disk file" sense; they usually won't run on the different operating system). The System Utilities program also performs all the standard file and disk manipulations such as formatting and copying disks and copying, deleting, and renaming files.

What I like best about this program is the interface. It's finally a true assembly language "magic menu" with windows (notice what happens when you ask for help) that conforms to Apple's *Human Interface Guidelines for the Apple II*. Apple talks about its guidelines all the time in reference to the Xerox interface but has all but abandoned its own "magic menu" design; it's good to see it in use on an Apple product here. But it's too bad Apple's guidelines are frozen in the days of DOS 3.3 rather than ProDOS. These utilities include no support for subdirectories, other than memorizing them and typing them in. It's amazing that neither AppleWorks nor these utilities offer the choice of choosing subdirectories by pointing at them and pressing return. Does anyone at Apple use ProDOS subdirectories? Is anyone at Apple aware of how third-party programs such as *Copy II Plus* support subdirectories?

There are a few other things these system utilities don't do. The most important missing feature is the ability to convert files between ProDOS and Macintosh formats. Apple has written a Macintosh program called Passport that will do this, but there are far more Apple IIs than Macintoshes in the world—an Apple II version makes more sense for a utility like this. The

program also lacks the ability to format DOS 3.3 disks, which earlier versions could do.

The new IIgs system disk also includes a new program called FASTCOPY.SYSTEM that uses IIgs and standard-slot RAM for copying disks. It's faster and requires many fewer disk swaps than anything Apple has offered before, although products from Glen Bredon and Bill Basham have been doing this for months already. (If you've ever tried to copy a 3.5 disk with Apple's older utilities and a single drive, you'll understand where this program is coming from—you have to swap disks literally dozens of times.)

If you have a IIe or IIc (or an 80-column 64K II-Plus, for that matter), you'll find the new system utilities run on those computers too, although the II-Plus and unenhanced IIes will show strange characters where there should be MouseText. IIe and IIc buyers nowadays get a system disk with their machines that includes version 2.1.1 of the system utilities.

The BASIC.SYSTEM that comes on the new IIgs system disk is still version 1.1. The date shown in the directory listing indicates how long it's been since an update. This is the program that would execute if there were no START file in the SYSTEM subdirectory.

BASIC.LAUNCHER is part of one of the world's few programs that takes ten times longer to load than to run. I cannot tell you the exact function of this file. If you try to run it it tells you not to do that. I assume it is used by LAUNCHER, which is in the SYSTEM subdirectory.

The APPLETALK subdirectory holds a number of files that you'll need if you want to use AppleTalk on your IIgs. Apple has still released no AppleTalk documentation for the Apple II that I can find, however, Don Lancaster has been taking IIgs AppleTalk apart and writing up what he's found in his "Ask the Guru" column in *Computer Shopper*. The December 1987 issue shows how to build the $130 AppleTalk cables for about $10 and the January 1988 issue shows how to send PostScript commands to the LaserWriter over AppleTalk, among other things. If you're using AppleTalk or LaserWriters, you should be reading Lancaster's column.

The ICONS subdirectory on the new system disk holds graphic images for various kinds of files. These images appear on your screen when you use a program called FINDER, which we'll look at later. Some commercial software comes with its own icon files; for best results, move these into the ICON subdirectory on your SYSTEM.DISK.

Logically, both APPLETALK and ICONS should be in the SYSTEM subdirectory, rather than cluttering up the root directory. I haven't yet had time to figure out if they work correctly if moved, however. The root directory of a ProDOS disk has a maximum capacity of 51 files. Subdirectories can officially have any number of files (although many programs, such as AppleWorks and FINDER, have definite limits to the number of files they can handle in a single subdirectory.) Whoever decided to put these files in the root directory isn't showing enough consideration for 20-megabyte hard disk users, who really need to have nothing but PRODOS, SYSTEM, and their own subdirectories in the root directory.

DIALOG.ICONS is a rogue file. No one seems to know exactly what it's for or why it's in the root directory. COPY.ME is a file that Apple's system disk tutorials use. It does nothing and has no other purpose. FINDER.DATA and FINDER.ROOT are data files created by the FINDER program to remember how you arranged, colored, and positioned this disk's icons the last time you had it on the FINDER desktop. These two don't appear on a virgin /SYSTEM.DISK, but will appear as soon as you manipulate the disk with *Finder*.

Now let's look in the SYSTEM subdirectory:

```
Filename          Blocks Type   Date

SYSTEM               1  DIR  23-SEP-87
..P8                32  SYS  17-APR-87  ProDOS 8, V 1.4
..P16               75  $F9  14-JUL-87  ProDOS 16, V 1.3
..START              1  $B3  13-JUL-87  boot-up P16 application program
..SYSTEM.SETUP       1  DIR  27-AUG-87  all files in this dir execute at boot
....TOOL.SETUP      68  $B6  15-JUL-87    required
....ATINIT          16  $E2   8-MAY-87    required with AppleTalk only
....ATLOAD.0         1  $B6   8-MAY-87    required with AppleTalk only
....SOUND.INIT       1  $B6  27-AUG-87    required
..TOOLS              1  DIR  27-AUG-87  this directory holds RAM-based tools
..DESK.ACCS          1  DIR  23-JUL-87  no desk accessories are supplied
..DRIVERS            1  DIR  23-SEP-87  P16 programs need these to print
....APPLETALK        6  $BB  12-JUL-87    required with AppleTalk only
....IMAGEWRITER     49  $BB  13-JUL-87
....LASERPREP       56  TXT  20-FEB-87    required with LaserWriter only
....LASERWRITER     53  $BB  12-JUL-87    required with LaserWriter only
....MODEM            5  $BB  13-MAY-87
....PRINTER          5  $BB  13-MAY-87
....PRINTER.SETUP    1  BIN  23-SEP-87
..FONTS              1  DIR  23-JUL-87  fonts for P16 programs
..FINDER           102  $B3  22-JUL-87  Macintosh emulator, needs 512K IIgs
..LAUNCHER          12  $B3  12-JUL-87  world's slowest program selector
..CLIPBOARD          1  $00  23-SEP-87  disk-based clipboard for P16 programs
```

P8 and P16 are the true ProDOS 8 and ProDOS 16 files. On the new system disk 3.1, ProDOS 8 is version 1.4 and ProDOS 16 is version 1.3. P8 is the file you use to replace PRODOS on IIe and IIc ProDOS disks so that they'll use the IIgs clock. Delete PRODOS from a *copy* of your older disk, copy P8 onto it, and rename P8 as PRODOS.

START is the program that runs when the disk is booted. It can be any renamed ProDOS 16 application program. The START included on the new system disk 3.1 figures out how much memory is available on your IIgs. If it can find at least 512K, it will run FINDER. If not, it runs LAUNCHER. If you use Glen Bredon's *ProSEL*, you'll want to replace this START file with the one on his /EXTRAS disk (if you have an older version of *ProSEL* without START and need an update, they're $5 from Bredon at 521 State Road, Princeton, NJ 08540; if you don't have it at all, get it—a new copy is $40).

SYSTEM.SETUP is the subdirectory that holds files that are executed when the system is booted. The PRODOS file handles this, as mentioned earlier. Only files of certain types are executed, primarily types $B6 and $B7. According to the ProDOS 16 Reference Manual, types $B8 (new desk accessories) and $B9 (classic desk accessories) will also be executed if found in this subdirectory, however, according to Apple Developer Technical Support, the manual shouldn't have said that. Desk accessories are supposed to be in the DESK.ACCS subdirectory, not SYSTEM.SETUP. On system disk 3.1, four files are found in SYSTEM.SETUP. Two of them, ATLOAD.0 and ATINIT, initialize AppleTalk.

There is also a new public domain program available to add SYSTEM.SETUP abilities to ProDOS 8. It was written by ***Open-Apple*** subscriber Sean Nolan and is called SETUP.SYSTEM. This program should be the first system file on your ProDOS 8 disks. If it is, ProDOS 8 will run it when you boot and it will look for a subdirectory called SETUPS and run all the SYS and BIN files it finds there. Then it will go back to the main directory and start up the next SYSTEM file. For more about Nolan's program, see his article in the November 1987 *Call-A.P.P.L.E.*, pages 14-23, and Bob Sander-Cederlof's article in the October 1987 *Apple Assembly Line*, pages 29-32. Sander-Cederlof also discovered a new secret feature of ProDOS 8, versions 1.3 and later—they will search the root directory for a type $E2 file named ATINIT and run it before running the first .SYSTEM file. ATINIT, of course, is the AppleTalk initialization file found in the SYSTEM.SETUP subdirectory of ProDOS 16 disks. Nolan's solution for running programs like ATINIT under ProDOS 8 is much more elegant than Apple's solution, but look for Apple's not-invented-here syndrome to prevent it from being used on future ProDOS 8 system disks.

The TOOLS, DESK.ACCS, DRIVERS, and FONTS subdirectories in the ProDOS 16 SYSTEM subdirectory all hold files that are used by ProDOS 16 application programs. Both TOOLS and FONTS hold a number of files on system disk 3.1—I haven't listed them here. DESK.ACCS, on the other hand, is empty. Those of you who are used to the Macintosh may be surprised to learn that there is no need for a FONT/DA MOVER on the IIgs. You simply use the FINDER or any other file manipulation program to move the fonts and desk accessories you want into the FONTS and DESK.ACCS subdirectories on a system disk and boot that disk to make it *the* System Disk. Those fonts and desk accessories will be available to your ProDOS 16 programs. On the other hand, note that fonts or desk accessories that are not on the System Disk (again, the disk you booted from), *are not* available to your ProDOS 16 programs.

As mentioned earlier, the root directory file ICONS holds logically the same kind of stuff as FONTS and DESK.ACCS and should have been placed in the SYSTEM subdirectory. APPLETALK, which holds slightly different stuff, could go here too.

The final three files in the SYSTEM subdirectory are FINDER, LAUNCHER, and CLIPBOARD. CLIPBOARD is a disk-based data file for ProDOS 16 programs that support a Macintosh-like clipboard. LAUNCHER is the kill-a-fly-with-a-sledgehammer program selector you'll get if you boot Apple's /SYSTEM.DISK on a IIgs with less than 512K. Replace it as soon as you can with *ProSEL*, or add more memory so you can run FINDER.

FINDER is a Macintosh desktop emulator. It allows you to copy, delete, rename, and startup files. It takes forever to start from a 3.5 inch disk, and apparently has some problems with hard disks larger than 10 megabytes, but otherwise is a pretty decent program for novice users. Compared to the

Macintosh, the big difference is that this *Finder* is in color. RAMdisks show up as little green logic boards. You can easily color-code your file folders (the icon for a subdirectory) if you want to. While the disk utilities that come with *ProSEL* and *Copy II Plus* are still my first and second choices, there have been times when what I wanted to do was easier with *Finder*; it's earned a place on my System Disk.

What you put on your own System Disk is up to you, but it should be obvious by now that you don't actually need *all* the files on /SYSTEM.DISK. If you don't use AppleTalk, you don't need the APPLETALK subdirectory or any of the files in it, you don't need the two AT files in SYSTEM.SETUP, and you don't need the APPLETALK file in DRIVERS. If you don't use a LaserWriter, you can also do away with LASERPREP and LASERWRITER from the DRIVERS subdirectory. By all means get rid of LAUNCHER in the SYSTEM subdirectory and BASIC.LAUNCHER in the root directory. COPY.ME should be deleted, of course, and SYS.UTILS and all its contents can be moved to an applications disk; there's no reason to have them on the System Disk.

What you don't want to delete are any of the files in TOOLS. When the next IIgs system disk comes out, you'll want to replace any updated tool files with the newer versions. You'll probably want to add files to FONTS, DESK.ACCS, and ICONS. Get new files for these subdirectories from commercial and public domain disks.

If you are a *ProSEL* user, remember to replace START with the *ProSEL* version. If you are also a RAMdisk user, proceed like this. First, run your favorite program for initializing disks and format both a new 3.5 disk and your RAMdisk. Name the 3.5 something like /COLDSTART and the RAMdisk something like /RAM5. Without a separate formatting step, Apple's standard slot RAMdisks won't boot.

Next, remove the 3.5 and copy all the files you want on your System Disk to the RAMdisk. Besides the core programs from Apple's /SYSTEM.DISK, you'll find you have room for AppleWorks, the *ProSEL* utilities, and whatever else you use most often. But don't use more than 1,525 blocks (about 760K).

Next, reinsert /COLDSTART and copy P8 from the /SYSTEM.DISK and RESTORE from your *ProSEL* disk onto it. Rename P8 as PRODOS; rename RESTORE as RESTORE.SYSTEM. Next run the *ProSEL* program BACKUP. When it asks you for the slot of the destination disk, enter a 0. It will then request a filename. Enter something like /COLDSTART/RAM.IMAGE. The program will then copy the entire contents of your RAM-based System Disk into a single, special file on the 3.5.

Once all this is done, run the *ProSEL* program BLOCK.WARDEN. Enter P for prefix and /COLDSTART. Enter F for file and RESTORE.SYSTEM. Enter E for edit and use the arrows to move the cursor to byte 6. A special little window will appear. Enter the name of the file your RAMdisk backup is in, RAM.IMAGE. Next, go to byte 49 and hit the tab key (tab allows you to enter ASCII data). Enter the name of the program you want to run when RESTORE is finished; probably /RAM5/PRODOS. Hit the tab key again (back to hex data) and go back to byte 48 and enter the length of this string in hex, 0C in this case. Press Escape to leave edit mode, W to write these changes onto the disk, and Q to quit.

You're done. Tomorrow morning turn on your computer and boot /COLDSTART. It will rebuild your System Disk in RAM and startup ProDOS 16 in not much more time than it takes to start ProDOS 16 from the 3.5 to start with. Put /COLDSTART away and spend the rest of the day with immediate access to the programs in your RAMdisk (even *Finder* takes "only" about 10 seconds to load). Use your 3.5 drive for data disks. Write and tell us how you like it.

Using the Advanced Interface

Back in October, in the midst of our "Far More Than You Ever Wanted to Know About Apple II Character-Oriented Interfaces" series, I mentioned that I'd include a longer assembly language example of how to use the Advanced Firmware Protocol (also known as Pascal 1.1 Firmware Protocol) "next month (or so)." Here it is, brief as I can make it.

The advantage of using the Advanced Firmware (rather than the Basic Firmware), is that it has a STATUS command you can use to find out if a character has been received from an input device (modem/keyboard) or if an output device (modem/printer) is ready for the next character. In addition, it has an INIT command you can execute without doing either a READ or WRITE. The Basic Interface has only READ and WRITE commands; the first READ or WRITE after PR# or IN# always initializes the firmware.

(The advanced interface INIT command should be used *one time* by your application program for each device you want to use. Its function is to reset the interface to its default dipswitch/auxmem/control panel settings (see "Modes and defaults," page 3.75). The advanced interface does this even on the Super Serial Card (you might pencil this in about half-way down the first column on page 3.76—the problems with Super Serial Card initialization identified there, it turns out, apply only to the Basic interface and can be easily solved with an INIT call to the advanced interface.) To deselect a device, simply stop using it. There is no need to use INIT more than once unless you have some reason to set the device to its default values more than once.)

(The advanced interface "ready for output?" status call can be used to determine whether a printer that is turned on is "selected" or not. However, if a printer is completely turned off, this call returns with a "yes—ready" response. This is because a low voltage is used by printers to signal when they are ready—if the printer is turned off the voltage on this wire will definitely be low. This is a hardware design flaw; it's not the fault of the firmware.)

The difficulty of using the Advanced Firmware is that Applesoft doesn't support it. You can use it directly from assembly language, but there are so many combinations of slots, commands, entry point offsets, entry points, and register initializations that things can quickly progress past the point that anyone can understand them.

For the sake of speed, it seems as if the way to proceed is to look up the entry point for each command/slot combination in advance, put the entry points for each slot in a table along with the proper X and Y register initialization values, then pull the proper stuff out of the tables just before each call. However, the entry points are *already in tables*, and pulling them out in advance is a waste of time. Instead, try the following routine. It's both fast and short. Call it with your registers loaded as follows:

```
Register contents when calling CHAR.IO

X = slot number of device (1-7 valid, no testing done).

Y = command (13-17 valid, no testing done).
        13 = INIT
        14 = READ
        15 = WRITE
        16 = CHARACTER RECEIVED?
        17 = READY FOR OUTPUT?

A = character to write (WRITE command only)
```

When control returns to you, the registers will hold the following information:

```
Register contents passed back by CHAR.IO

X = error code
        0 = no error
        3 = illegal operation
    32-47 = Super Serial Card communications errors (see October, page 3.68)
       64 = device error

Y = undefined
A = character read (READ command only)
carry = answer to STATUS commands 16 & 17 (1=Y, 0=N)
```

The routine requires two zero-page bytes for temporary manipulations. As written, it uses bytes 6 and 7, which don't conflict with Applesoft, the Monitor, DOS 3.3, or ProDOS, but which could conflict with other languages or other programs:

```
*---------------------------------------------------------------
*         : CHAR.IO
*         :    directs calls to Advanced Character I/O Firmware
*         :
*         : by Tom Weishaar, December 1987
*---------------------------------------------------------------

            ZP.TEMP .EQ 6       We need a two-byte pointer on page zero.
            MSLOT   .EQ $7F8    Interrupt handlers expect $Cs here.

            CHAR.IO
0300:C0 10     CPY #16          First, see if command is 16 or 17
0302:90 09     BCC IO.ALT          if neither, continue at ALT entry
0304:F0 05     BEQ .1              if 16, continue at .1
0306:88        DEY                 if 17, decrement Y to 16 and
0307:A9 00     LDA #0                 put a 0 in A (ready for output?)
0309:F0 02     BEQ IO.ALT             and continue at IO.ALT
030B:A9 01  .1 LDA #1              if 16, put 1 in A (character ready?)
            IO.ALT
030D:48        PHA              Save A on stack
030E:8A        TXA              Using slot number that was passed in X,
030F:0A        ASL                 create $s0 for later, and...
```

```
0310:0A           ASL
0311:0A           ASL
0312:0A           ASL
0313:48           PHA               save it on the stack.
0314:8A           TXA             Using slot number that was passed in X,
0315:09 C0        ORA #$C0          create $Cs, and...
0317:AA           TAX                  save it in X, and...
0318:86 07        STX ZP.TEMP+1        put it in our pointer.
031A:A9 00        LDA #0
031C:85 06        STA ZP.TEMP     Pointer now holds $Cs00.
031E:B1 06        LDA (ZP.TEMP),Y Using offset from Y, get entry point XX.
0320:85 06        STA ZP.TEMP     Pointer now holds $CsXX.
0322:68           PLA             Get s0 and...
0323:A8           TAY                  put it in Y.
0324:68           PLA             Get original contents of A.
0325:8E F8 07     STX MSLOT       Fix MSLOT for interrupt handlers.
0328:8E FF CF     STX $CFFF       Turn off all $C800 ROMs (see p 3.67).
032B:6C 06 00     JMP (ZP.TEMP)   Jump to slot/command entry point.
```

This 46-byte routine takes care of finding the correct entry point for the slot and command you want, of loading the registers with the required values, and of taking care of a couple of Super Serial Card bugs mentioned in the October issue near the end of page 3.67. It is also position-independent—it works anywhere. You can make it slightly faster by including tables of X and Y register values, but the speed advantage is slight (5 machine cycles), 12 extra bytes of space are required, and it will no longer be position-independent. If you're still interested, add these tables after the JMP (ZP.TEMP) and change the section from $30E to $316 to:

```
            LDA SLOT0.TABLE-1,X   Get s0 value for this slot and...
            PHA                      save for later.
            LDA CSLOT.TABLE-1,X   Get Cs value for this slot.

         SLOT0.TABLE
            .HS 10.20.30.40.50.60.70
         CSLOT.TABLE
            .HS C1.C2.C3.C4.C5.C6.C7
```

There are a number of ways to use CHAR.IO from an assembly language program. For example, if you don't mind having just one current SLOT value, like Applesoft does, add a section of (C)haracter I/O commands that looks like this:

```
         C.INIT
            LDY #13
            BNE CHAR.GO
         C.READ
            LDY #14
            BNE CHAR.GO
         C.WRITE
            LDY #15
            BNE CHAR.G0
         C.READ.TEST
            LDY #16
            BNE CHAR.GO
         C.WRITE.TEST
            LDY #17
         CHAR.GO
            LDX SLOT
            JMP CHAR.IO
```

Now all that would remain is for the programmer to load the A register with a character before doing a WRITE, looking in the A register for a character after a READ, and branching off the carry for STATUS results (BCS = ready, BCC = not ready). It also might be interesting to look in the X register for an error, especially after INIT calls, but the lack of support for X register errors (see page 3.68) generally makes this a waste of time. For example:

```
to init slot 1    LDA #1
                  STA SLOT          Specify slot and...
                  JSR C.INIT           initialize a character device
                  TXA               Test for error...
                  BNE ERROR            problems with initialization.

to read           JSR C.READ        Character returned in A register.

printer ready?    JSR C.WRITE.TEST
                  BCS READY         Printer online, paper ok
```

The final possibility we'll discuss here is using the Advanced Interface from Applesoft. As mentioned in October (page 3.66), there are rumors of incompatibilities between Applesoft and at least some Advanced Interface firmware, but the only conflict we've identified so far is limited to mixing the Advanced Interface with Applesoft low-resolution graphic commands. If anyone finds anything more specific by using the following demo program, please let us know.

The difficulty of using a routine like CHAR.IO from Applesoft is that Applesoft has no commands for "register loading." You can CALL an assembly language routine from Applesoft, but you can't specify register contents before the call and you can't determine register contents after the call. So we have to write an assembly language "front end" for CHAR.IO that includes the ability to do these things for us. Then, from Applesoft, we'll POKE the register values we want to memory, CALL the new routine, and find what values were returned with PEEKs.

Another difficulty of using CHAR.IO from a higher level language like Applesoft is that CHAR.IO is written as a fast, "low level," assembly language routine and consequently does *no error checking*. It will accept a slot number that doesn't support the Advanced Firmware or even a slot number that doesn't exist. It will dutifully execute command #133 if you tell it to, even though there is no such command. In all such cases your computer will crash, lock-up, or worse. Error checking, if there is to be any, will have to be done by our "front end."

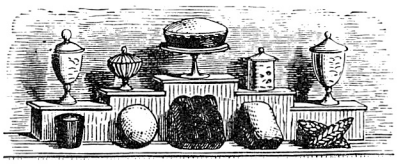
This brings up the question of how one determines whether the device in any particular slot supports the Advanced Firmware protocol. Back in October (page 3.66) we discussed the identification protocol for advanced firmware; the following "front end" includes a new command, TEST (command 18), that determines whether a slot holds an advanced firmware device and, if it does, returns its identification byte:

```
         *--------------------------------------------------------------
         *          : CHAR.FP
         *          :    Applesoft ''front end'' for CHAR.IO
         *          :
         *          : by Tom Weishaar, December 1987
         *--------------------------------------------------------------

                  CHAR.FP
032E:A5 06           LDA ZP.TEMP      Save the contents of our temporary
0330:48              PHA                 zero-page bytes for safety's sake.
0331:A5 07           LDA ZP.TEMP+1
0333:48              PHA
0334:AE 96 03        LDX SLOT         Load slot number into X,
0337:F0 11           BEQ IO.ERR          if 0, report an error
0339:E0 08           CPX #8              if 8 or more,
033B:B0 0D           BCS IO.ERR             report an error.
033D:AC 95 03        LDY CMD          Load command number into Y,
0340:C0 0D           CPY #13             if less than 13,
0342:90 06           BCC IO.ERR             report an error.
0344:C0 12           CPY #18             if 13 to 18, cmd is ok,
0346:F0 06           BEQ TEST            branch for command 18
0348:90 32           BCC FE.2            branch for commands 13-17.
                  IO.ERR
034A:A2 03           LDX #3           Load Illegal Operation error into X
034C:D0 3D           BNE IO.OUT          and return to caller.
                  TEST
034E:8A              TXA              Test slot firmware to see if it
034F:09 C0           ORA #$C0            supports the advanced interface.
0351:85 07           STA ZP.TEMP+1
0353:A9 00           LDA #0
0355:85 06           STA ZP.TEMP      put $Cs00 in pointer.
0357:A2 05           LDX #5
0359:BC 76 03     .1 LDY TABLE,X      get offset from table.
035C:CA              DEX
035D:B1 06           LDA (ZP.TEMP),Y  get value at this offset.
035F:DD 76 03        CMP TABLE,X      compare to correct id, not equal
0362:D0 E6           BNE IO.ERR          means no support, send back error #3
0364:CA              DEX
0365:10 F2           BPL .1           try next offset until all pass test
0367:A0 0C           LDY #$C
0369:B1 06           LDA (ZP.TEMP),Y  get device signature byte
036B:8D 98 03        STA READY        put whole thing in READY
036E:29 F0           AND #$F0         clear second hex digit
0370:8D 97 03        STA CHAR         put what's left in CHAR
0373:E8              INX              make X=0 for no error
0374:F0 15           BEQ IO.OUT
                  TABLE
                     .HS 38.05.18.07.01.0B
0376:38 05 18 07 01 0B
```

```
                FE.2
037C:AD 97 03       LDA CHAR          Put character in A.
037F:20 00 03       JSR CHAR.IO       Use CHAR.IO to route slot/command/char.
0382:8D 97 03       STA CHAR          Save A in CHAR.
0385:A9 00          LDA #0            Save carry in READY.
0387:2A             ROL
0388:8D 98 03       STA READY
                IO.OUT
038B:8E 99 03       STX ERROR         Save X in ERROR.
038E:68             PLA               Restore zero-page temporary location.
038F:85 07          STA ZP.TEMP+1
0391:68             PLA
0392:85 06          STA ZP.TEMP
0394:60             RTS               Done.

0395:00         CMD   .DA #00         PEEK/POKE 917
0396:00         SLOT  .DA #00         PEEK/POKE 918
0397:00         CHAR  .DA #00         PEEK/POKE 919
0398:00         READY .DA #00         PEEK/POKE 920
0399:00         ERROR .DA #00         PEEK/POKE 921
```

To play with these routines from Applesoft, you need to assemble them or hand-enter the hex codes. I tried both and was able to hand-enter them from the Monitor faster than I could find the disk my assembler is on. Most Apple IIs nowadays also have a built-in Mini-Assembler (see May 1987, page 3.27) that you could use to enter the code. BSAVE the results in a file called CHAR.IO (A$300,L$9A). Then write yourself some playful Applesoft programs such as the following one:

```
10 HOME
20 PRINT  CHR$ (4);"BLOAD CHAR.IO"
25 CMD  = 917 : SLOT = 918 : CHAR = 919 : REDY = 920 : ERR  = 921
30 DIM A$(16)
40 A$(0) = "reserved"
41 A$(1) = "printer card"
42 A$(2) = "mouse/joystick card"
43 A$(3) = "serial/parallel card"
44 A$(4) = "internal modem"
45 A$(5) = "sound/speech card"
46 A$(6) = "clock card"
47 A$(7) = "mass storage device"
48 A$(8) = "80-column card"
49 A$(9) = "network interface"
50 FOR I=10 TO 15 : A$(I) = "unknown" : NEXT
100 POKE CMD,18             : REM scan slots and TEST for advanced firmware
110 FOR S = 1 TO 7
120 : POKE SLOT,S           : REM CMD and SLOT must be specified before CALL
130 : CALL 814              : REM call CHAR.FP
140 : IF PEEK(ERR) THEN 160 : REM if ERR < > 0 then no advanced firmware
150 : PRINT "The device is slot #";S;" is a ";A$(PEEK(CHAR) / 16);"."
160 NEXT
190 END
```



Miscellanea

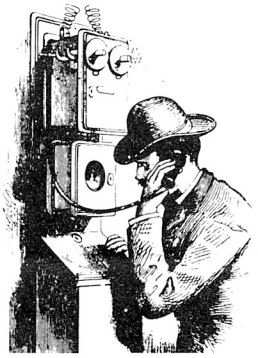
A year ago this month we were talking about the Smartport *block-oriented* firmware used by the Apple II for disk storage (contrast with the *character-oriented* firmware used for printers and similar devices). One of the commands we discussed, (page 2.92) was a status call to the Smartport itself, which is supposed to return the number of devices connected to the Smartport. It turns out that on the original version of the memory-expandable IIc, making this call to the slot 4 firmware implied a RAMdisk was present whether the machine had a memory card plugged in or not.

This has been fixed in the "revised expandable IIc." You can identify these machines with a PEEK(64447) ($FBBF). The original IIc had an $FF here, the original 3.5 ROM version a $00, the original memory expansion version a $03, and the current version a $04. You might want to pencil this into our "Apple II Family Identification Bytes" table on page 2.66.

To figure out if there really is a card plugged into a IIc you should do a Smartport status call to the *device* (rather than to the Smartport, see page 2.92 for complete details) and check how many blocks of storage are available. If no card is plugged in, this field will come back filled with goose eggs.

The latest revision of the IIc ROM also fixes a problem with keyboard buffering. Apple's *Apple IIc Technical Notes #5, #6,* and #7 have complete details.

Our friends at Beagle Bros have moved. Their new address is 6215 Ferris Square, Suite 100, San Deigo, CA 92121. Their new phone numbers are 619-452-5500 (general), 619-452-5502 (technical support), and 619-452-5565 (bulletin board: 300/1200/2400, 8N1, full duplex).



Ask (or tell) Uncle DOS

More feedback on reality

In the midst of the December letter from Ben Kobb (page 3.84) you say you can't believe that Apple's Jean-Louis Gassee would say, "in no way, shape or form will we be in the office with this computer" (the IIgs). I don't know whether any such statement was made; BUT I do know that an interview of Gassee published in Waldenbooks' *Computer NewsLink* this fall certainly inferred that only the Macintosh was meant for business.

And here I sit, typing this note on AppleWorks and the IIe I have used for business since March 1983. I have also purchased a IIgs and expected great things, 16 bits and whatever. Well, as you might guess, the best programs I have are AppleWorks, *Apple Writer,* and *ProSEL.*

Jim DeMoss
Grand Blanc, Mich.

As a professional journalist, I should warn you that written interviews often reflect the feelings of the interviewer as much or more than the feelings of the interviewee....As promised, I contacted Gassee on AppleLink and asked him to confirm or deny the quotation and tell us why or why not he thinks the Apple II has a place in the office. Here's his response:

I think the Apple IIgs has a privileged place in the school, the office, and at home.

Jean-Louis Gassee
Apple Computer, Inc.

While his answer isn't all I would like it to be, it is something, and it is adequate. The people running Apple Inc are just that, people. They have been focused on their Macintosh line for three long years. Let's admit it, the Macintosh needed the attention. Without it, it would have gone the way of the Apple III and the Lisa.

But now that the Macintosh can stand on its own feet, and Apple's managers have the second successful computer they've been working so long to develop, it's time for them to give the II line the attention and the tender loving marketing it deserves. Apple is simply not tapping the potential of the II family. Our jobs here are to open their minds to the possibilities. Apple's managers are intelligent, motivated people who will respond to our arguments if we can persuade them of the truth we see.

Uncle DOS has received a number of other letters from subscribers on these issues. The primary concerns are that Apple purposefully crippled the speed of the IIgs microprocessor to avoid competing with the Macintosh, that 16 bit IIgs software runs unacceptably slow, that the cost of the Apple II is too high when compared to MS-DOS machines, that comparatively little new software is being developed for the Apple II compared to MS-DOS and the Macintosh, and that Apple's Macintosh marketing has done more to convince people that the Apple II isn't suitable for business than it has done to convince them that the Macintosh is.

The unusual thing is that when companies have products that are competing with each other, it is usually the established product that gets preference and the new product that is crippled. I have talked to the engineers who designed the IIgs; I'm convinced that any speed limits they put on the machine are there because they didn't want to drive up the cost, not because they didn't want to scare their Macintosh colleagues. If Apple was worried about the IIgs competing with the Macintosh, you can be sure the IIgs wouldn't have memory expansion capabilities well beyond those of most Macintoshes.

The problem is that Apple's managers seem bent on turning the Apple II into a cash cow—a mature product that produces money to be invested elsewhere—before its time. If Apple would put the kind of resources into getting IIgs software published that it has put into Macintosh software for the last three years, if Apple would resume marketing the Apple II to adults as well as children, and if Apple would adjust its prices to meet the MS-DOS competition, most of you would be happy. If you were

happy, you would sell Apple IIs by word-of-mouth; something, it is apparent from your letters, many of you have stopped doing.

Apple is not a monolith. It is not a stone rolled by an iron hand. It's people who respond to persuasion and the facts. We need to convince them that their current neglect is wasting the potential of the Apple II.

Warm French dough

Include AppleWorks with the Apple II? This has been Apple's practice in Europe for some time as far as the IIe and IIc go. Most people buy the packages, as buying individual components is much more expensive. Carrying this approach further, the standard Apple IIc in France now comes with the extra 256K memory card, yet is priced below what the IIc sold for without the card.

Apple France seems to recognize that Apple II users now want that memory, and it's not for playing games or anything other than AppleWorks. As Jean-Louis Gassee certainly knows, having played the major role in Apple's success in France, the Apple II has a significant small business and home office role here, and Apple marketing has not ignored that potential.

Now for the bad news. Apple France seems worried that AppleWorks isn't a 16 bit program. To show off the IIgs they have bundled it with a painfully slow word processor, *GS Write*, along with a better paint program, *GS Paint*. It is nice to get such software thrown into the package and the IIgs has been selling very well, but the real lucky people were those of us who bought the IIgs before *GS Write* was ready; we were given AppleWorks at no extra charge to tide us over. I expect most, like I, after a little playing with the mouse and cute fonts, are back to using AppleWorks regularly and praying someone will listen to Frank Lowney's plea for a 16-bit version (December, pages 3.83-84).

In sum, the Apple marketing and customer support I find here in France is far superior to what I find in the States. Apples are not cheap here, but the well-developed product and company image makes consumers willing to pay the premium. When they go into most dealers, customers are presented with well-packaged systems, and the Apple IIs aren't kept in the background. I really wish the parent company would learn a few of the tricks developed by its French offspring.

Bill Witherell
Paris, France

Everything in its place

Frank Lowney's letter was exactly on target. Like him, I own a number of the latest 16 bit programs like *MultiScribe GS*, yet AppleWorks is far and away my first choice. AppleWorks is on my screen in seconds, completely ready with enhancements from Beagle Bros, Pinpoint, and Applied Engineering; *MultiScribe GS* takes minutes. Getting *MacWrite* up and going takes an intermediate amount of time. Using *MacWrite* without keyboard cursor control, however, is a pain in the chips. Your comments hit the mark precisely, too: why not *faster* 16 bit software instead of bells and whistles.

I teach using computers at a secondary school. In a twist of perceived convention, I use Macs to teach simple junk but use Apple IIs in the advanced classes: for programming, graphics, robotics, engineering-related subjects, and so on. Because of its open architecture, the Apple II is a much more powerful machine than the Mac for our rather general purposes. Nearly all the speed of the Mac is sacrificed to enormous bit-mapped take-along files that the computer must fiddle with; the result is that for most actual uses, the Mac is no faster than the II. The Mac is wonderful, no question — but so is the Apple II.

Michael Simonds
San Diego, Calif.

AppleWorks 16

In my letter published in last month's issue, I suggested we begin a realistic discussion of what a 16-bit AppleWorks could and should do. In order to practice what I've preached, let me contribute at least one idea to the cause.

The real genius of the AppleWorks filecard interface is that each screen communicates so much information in the context of what you are currently attempting to do. That is, you know where you've been, you know where you are, and you know what options lie immediately ahead. Not only does this make the learning curve mercifully short and steep but it also invites exploration and, thus, innovation. But one wonders why this thinking was not extended to the way files are identified and brought to the desktop if the file you seek is not in the main directory or is an ASCII or DIF file.

AppleWorks now contravenes the purposes of the ProDOS hierarchical file system. Looking beyond the main directory involves getting into the AppleWorks variant of Blind-Man's-Bluff unless you have the assistance of a remarkable memory or Randy Brandt's *PathFinder*.

AppleWorks 16 will need to display the directory structure and allow the user to highlight any subdirectory to view or select files from. And what about the return trip? It is quite likely that you will have several files on your desktop from different parts of the hierarchical file structure. If AppleWorks would remember the origin of each file it could automatically save to the file's "home" subdirectory. Of course, it wouldn't be AppleWorks if it didn't ask "Save file to...?" At this point one could choose an alternate subdirectory from the directory structure display.

Is there a better way to do it? Are there other ideas for AppleWorks 16 out there? Let's talk about it.

Frank Lowney
Milledgeville, Ga.

End of the line?

After all the talk about a 16-bit AppleWorks in the December issue, I felt compelled to write and contribute my 8 bits.

It should be made clear that AppleWorks 2.0 already has 16-bit code. AppleWorks' memory management and some of the display routines are contained in what is called the SubHost. Several versions of the SubHost are on the AppleWorks startup disk — SEG.00 is the standard one for 128K machines, SEG.XM is for Apple-style memory expansion cards, and SEG.RM is for the IIgs. SEG.RM is written in 65816 code, although that always seems to get ignored. Some 16-bit code is also found in APLWORKS.SYSTEM.

Free tip: you can save disk space by deleting SubHost files that don't apply to your hardware setup; this is especially handy when running AppleWorks off of RAMdisks.

Most of the desired enhancements you ask for at the end of page 3.84 are already in place. When run on a IIgs, AppleWorks is faster and allows a larger desktop, larger AWP files, and larger ADB files. If you make AppleWorks 2.0 think that it's running on a IIe instead of a IIgs using the patch on page 3.86, you lose out on the added speed of the 16-bit code and you're back to a 55K desktop and smaller file limits.

As for developer enhancing, the Beagle Bros *TimeOut Series* provides smoothly integrated applications such a graphing, spell-checking, a programming language, printing with Macintosh fonts, and much more. The beauty of *TimeOut* is that it even works with unenhanced IIes, and it's available now; there's no need to dream and wait for the future.

I'm all for bundling AppleWorks 2.0 with IIs, but I think that any developer who decides to write the longed for "16-bit AppleWorks" is going to write it using the IIgs graphics interface. The text version already exists in AppleWorks 2.0!

Randy Brandt
Beagle Bros, Inc.

*What Beagle Bros has done with AppleWorks 2.0 is, of course, the model for much of what we've been talking about the last couple of months. But you guys have developed your knowledge of AppleWorks the hard way, by disassembling it and figuring out how it works. It's news to all of us, for example, that there's 16-bit code in AppleWorks 2.0. Apple's press release didn't mention that, their advertising certainly hasn't bragged about it, their developer tech support people say it's not system software and they don't know anything about it, Claris says its not ready to support it, and it's **still** the best-selling program in the Apple II world.*

*I think we would all welcome an AppleWorks with a graphics interface **if** it was as fast as AppleWorks 2.0. The problem here is speed. If a graphics interface was as important as speed, **MouseCalc** and the rest of International Solution's software would still be with us. But speed is far more important, except for software that can only be done well with a graphics interface to begin with (drawing, graphing, desktop publishing).*

If AppleWorks 2.0 is the fastest software we're ever going to get, well, that's not too shabby, and, as you point out, it runs on unenhanced IIes. I find it hard to believe that spreadsheet recalculations and data base sorting and word processor replacing couldn't be markedly improved by rewriting it all in 65816 machine code, however. The problem is that Apple has managed to convince the people who have to sell software that only graphics-based programs are worth selling. Now if it could just convince the people who buy software that speed isn't important, all would be right with its vision of the world.

The reason for the IIgs defeater patch is to force AppleWorks (both 1.3 and 2.0) to expand into an Apple-style standard-slot memory card on the IIgs, which it will absolutely refuse to do otherwise. We appreciate knowing the RAMifications and that we can delete the two SEG files we don't need from our RAMdisks.

IIgs defeater defeated

I cannot help feeling that there is something wrong with your patch to make AppleWorks think a IIgs is a IIe (page 3.86). APLWORKS.SYSTEM is only 8,531 bytes long; in this light, your B parameter of 13605 seems a bit out of line.

You may be pleased to learn that I have urged all my friends to buy their own **Open-Apple** subscriptions (mainly so they will stop borrowing my copies.) This should mean another $100 or so for you if they carry out their threats.

Jerry E. Kindall
Grove City, Ohio

Yep, those B parameters are about $2000 too high. Not only that, one of them points to a JMP following the JSR $FE1F rather than to the JSR (it would still work, but it's not what we said to do). Get your pencil and change the B parameters shown on page 3.86 to:

```
B5031 (V 1.3)        B5408 (V 2.0)
```

We don't mind you loaning out your copies; what bothers us is people who make photocopies without paying the required fees (15 cents per page per photocopy).

More database crashes

Your latest newsletter arrived just before I was going to sit down and write you a letter about a new bug I found in AppleWorks 2.0. It appears that Eugene Whitehouse ("Database crash lead," December, page 3.87) has also had the same problem. I have some more information that may help. I now have two failure examples from the same large data base.

The first case had eight report formats. When I went through the data on the disk, the number-of-formats byte in the header still said eight, but there were only two formats on the disk. In the second case, the header said the file had eight formats, but there were only three. Something is causing report formats to be dropped in multiples of 600 bytes at a time. It appears to be a function of how full a disk is. I can save to a disk and get an unreadable copy when the disk is full with a previous copy of the file. If I reformat the disk and save the same file the copy is readable.

Open-Apple

[redacted]

is written, edited, published, and

© Copyright 1988 by Tom Weishaar

with help from

Tom Vanderpool **Sally Dwyer**
Dennis Doms **Steve Kelly**

Most rights reserved. All *programs* published in ***Open-Apple*** are public domain and may be copied and distributed without charge. Apple user groups and significant others may obtain permission to reprint *articles* from time to time by specific written request.

Open-Apple has been published monthly since January 1985. Worldwide prices (in U.S. dollars: airmail delivery included at no additional charge): $24 for 1 year; $44 for 2 years; $60 for 3 years. All back issues are currently available for $2 each: bound , indexed editions of our first three volumes 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 correspondence to:

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

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

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

WARRANTY AND LIMITATION OF LIABILITY. I warrant that most of the information in ***Open-Apple*** is useful and correct, although drivel and mistakes are included from time to time, usually unintentionally. Unsatisfied subscribers may cancel their subscription at any time and receive a full refund of their last subscription payment. The unfilled portion of any paid subscription will be refunded even to satisfied subscribers upon request. MY LIABILITY FOR ERRORS AND OMISSIONS IS LIMITED TO THIS PUBLICATION'S PURCHASE PRICE. In no case shall I or my contributors be liable for any incidental or consequential damages, nor for ANY damages in excess of the fees paid by a subscriber.

ISSN 0885-4017
Printed in the U.S.A.

I am mystified as to what is causing the problem. Curing the problem is as simple as identifying the number of formats that are really on the disk and correcting the number-of-format bytes to match. However, you do lose the missing report formats permanently.

Sidney A. Powers
Valencia, Calif.

We also have another letter from subscriber Brendan Hoar documenting the problem. It seems to happen only when a data base file is saved onto an almost full 5-1/4 floppy. One fact we can't sort out from the letters we've seen is whether Applied Engineering's AppleWorks expansion software is required to make the bug appear. If any of you are able to replicate the bug, please do some tests with and without modified software and let us know what happens.

IIgs Smartport bug

I recently discovered a bug in both the original and version 01 IIgs ROMs. The problem occurs when you make a Smartport call to the firmware in slot 5. The firmware destroys locations $57 through $5A on the caller's direct page. This only occurs with Smartport calls (i.e., calls to $C50D), not ProDOS calls (calls to $C50A). The old ROM sets the direct page to $0000 prior to executing the offending code; this results in $57 through $59 on the "true" zero-page being trashed. The new ROM does not change the direct page, so $57 through $5A on the user's direct page are destroyed.

It is ironic that just a few instructions after this code the firmware calls a routine that saves all the zero-page locations that it will be using.

Alan J. Silver
Yorba Linda, Calif.

Backup bit clearing

How are you supposed to clear the backup bit in a file's ProDOS directory entry? The *ProDOS Technical Reference Manual*, p. 172, states: "ProDOS sets bit 5, the backup bit, of the access field to 1 whenever the file is changed (that is, after a CREATE, RENAME, CLOSE after WRITE, or SET_FILE_INFO operation). This bit should be reset to 0 whenever the file is duplicated by a backup program. Note: Only ProDOS may change bits 2-4; only backup programs should clear bit 5, using SET_FILE_INFO."

According to the first sentence, calling SET_FILE_INFO will set the backup bit; the last sentence says that you should use SET_FILE_INFO to clear the backup bit. How can SET_FILE_INFO clear the backup bit if it always sets it?

Paul Lucas
Levittown, N.Y.

Subscriber Paul Santa-Maria asked this same question, but also pointed out that on page 94 of the manual, in the explanation of the System Global Page, there appears the following entry:

```
BF95: BUBIT DFB $00  ;controls use of backup bit
```

Dennis figured out that to clear the backup bit you need to store a $20 in BUBIT, then call SET_FILE_INFO. This bit of information seems to be missing from all versions of the ProDOS 8 technical manuals. ProDOS 16 has a whole new command for this, CLEAR_BACKUP_BIT. For more, see the October issue of ***Apple Assembly Line****, page 26. Santa-Maria sent the same question there and Bob Sander-Cederlof also figured out the answer, which he explains in more detail.*

Advanced interface bugs

During the past three years, we have written printer and interface drivers for *Time is Money* (Turning Point Software), *Sideways* (Funk Software), and *Publish-IT!* (Timeworks). When we write these drivers, we try to follow existing standards. Our specific intent is to send data through the interface without any translation by the interface card. In the good old days, when standards were largely ignored, we would write data directly to the hardware ports of the Super Serial Card.

When Apple released the IIc, which wasn't compatible on a hardware level with the Super Serial Card, we wrote a second hardware-level driver. For the IIgs we wrote a driver that uses the Pascal interface. We prefer the Pascal interface over the Basic interface for two reasons. First, we can initialize the card without sending any data to the output device (JSR $Cs00, the Basic interface initialization, always sends the character in the accumulator to the printer or other output device). Second, we can initialize the interface without the risk of hanging the computer. With the Basic interface, if the printer is off-line, the program will hang when you try to send data to it. With the Pascal interface, it is possible to check the status of the output port before sending data.

Unfortunately, the Pascal interface cannot be universally used because of bugs in the Super Serial Card and the original IIc ROMs.

Revision 1 IIc ROMs (the models that don't support 3.5 drives) will not correctly return status because of a missing CMP #$10 at $C227 in the firmware. The result is that the status may be okay, but the result of the status call says it isn't. This prevents using the Pascal interface with early IIcs.

With the Super Serial Card, if you print with one program using the Pascal protocol and then print with a second program without resetting or turning off the Apple, the second program will print without line feeds. When you initialize the interface via Pascal, automatic linefeeds are turned off. When you initialize the interface via Basic, automatic linefeeds are supposed to be turned on, but as you point out on page 3.76, the Basic firmware won't reinitialize itself except after power-up or reset.

The reason is that in the early days, some programmers sent all data to the printer by repeatedly calling $Cs00, rather than making a single call to $Cs00 and subsequent calls to address left in the output hooks at $36, which is the correct Basic protocol. In order to protect programmers who repeatedly called $Cs00, the Super Serial Card had to be designed so that it didn't really reinitialize itself after every call to $Cs00.

Bruce D. Rosenblum
Turning Point Software
Boston, Mass.

Repeatedly calling $Cs00 on the IIgs will cause it to repeatedly reinitialize itself (which makes interface command codes seem not to work), so it's time for programmers to learn how to correctly use either the basic interface or the advanced (Pascal) interface. It's possible to solve the second problem you identify by using the Super Serial Card Reset command as part of your quit routine. See page 3.76 for an example of this; unfortunately, you have to send the command through the basic interface; the advanced interface ignores this command. Thanks for warning us about the status bug in the early IIc. As you point out, it makes writing a single, universal interface driver difficult to impossible.