Open-Apple™

January 1987
Vol. 2, No. 12

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

Releasing the power to everyone.

How to keep from getting pregnant

Of all the literary works that I've collected, the most dog-eared is a small volume I've had more than 12 years and still refer to at least weekly. An early section of this work has a title very similar to the one on this article. That section begins like this:

Say you are given a new racing Ferrari that you will be forced to drive for the next seventy-five years. No doubt you'd clean the motor with Q-Tips and hover over its gas and oil mixtures like a French chef pouring aspic.

In exactly the same way, you are imprisoned in the vehicle of your body (for seventy-five years!) with a chassis that can never be exchanged nor replaced. Poor fuel reduces your speed because it starves your body, mind, and spirit. Don't blame your psychological problems **all** *on early home life and your parents. You may be creating your own problems right now—or at least not actively correcting them—by constant consumption of Non-Foods, those over-processed, over-priced products that litter our grocery shelves....*

The book, known world-wide as "The Campus Survival Cookbook," by Jacqueline Wood and Joelyn Scott Gilchrist, goes on to chop and slice home the wisdom of eating a decent breakfast.

The book comes to mind because here at ***Open-Apple*** you can be sure we'd never begin an article with a title like this just to trick you into reading it. Before we get down to the bare facts, however, I was wondering whether you II-Plus and IIe users knew that you can easily add a connector to your computer that's much like the disk connector found on the back of the IIc and IIgs?

An unlimited number of slots. This little connector has all kinds of potential uses. Apple's technical manuals call it the Smartport Bus connector. Theoretically, you could connect all kinds of devices to your Apple II through this connector—printers, modems, robots, and compact disk players, to name a few. What's neat is that all of these devices could be connected to your computer *at the same time* and use up just one slot. This is because Smartport Bus-compatible devices can be daisy-chained together—each is supposed to have a connector on the back that the next device can be plugged into. (To avoid having your devices jerked all over your computer room, however, I recommend giving the robot plenty of cable and plugging him in at the end of the chain.)

Unfortunately, this is all theoretical—at the moment there aren't any printers, modems, or robots that will connect to a Smartport Bus. The only things you can successfully connect to the Smartport Bus today are disk drives. We'll look at the kinds of disk drives that work on the Smartport Bus in a moment—first though, let's examine two other ways of connecting daisy-chains of devices to Apple IIs.

One of these is the AppleTalk network, which is available only on the new IIgs. Much like the Smartport Bus, AppleTalk allows you to connect your computer to a chain of devices such as printers, modems, and disk drives. AppleTalk has the advantage of allowing more than one computer to be a part of the chain. Consequently, AppleTalk also has the disadvantage of making you share your devices with other people. Unlike the Smartport Bus, which is all theoretical except for disk drives, AppleTalk, at the moment, is all theoretical except for printers.

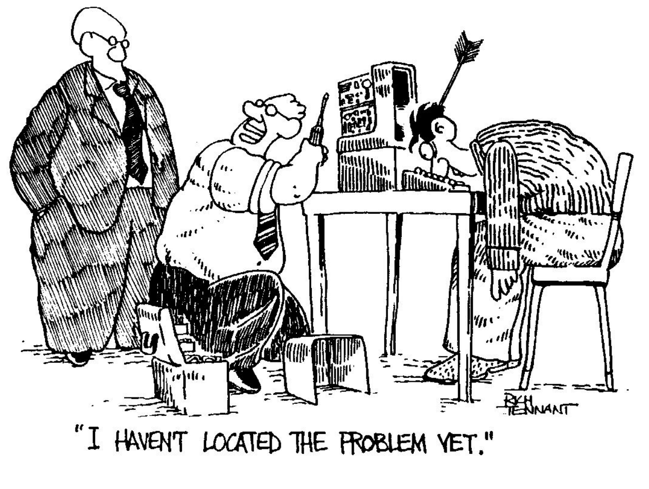
The other method of daisy-chaining devices is called the Small Computer System Interface (SCSI, pronounced "scuzzy"). Apple announced its SCSI controller card, which works in any slotted Apple II, including the II-Plus, in late September (see ***Open-Apple***, October 1986, page 2.67). Other vendors are announcing similar cards. The advantage of SCSI is that it is a computer-industry-wide standard. SCSI-compatible devices work with any computer that has a SCSI Bus connector. Consequently, SCSI devices should have bigger markets and lower prices than Smartport Bus devices, which will probably all be manufactured by Apple, Inc. The SCSI standard allows several different types of devices to be connected to the daisy chain, including direct-access devices, such as disk drives; sequential-access devices, such as tape drives; printers; "processor devices," such as other computers; write-once read-multiple (WORM) devices, such as optical disks; and read-only devices, such as compact disks (CD-ROM).

A small disadvantage of SCSI in the Apple II world is that it can only be used in slotted Apples. The Apple IIc can't use it. In addition, SCSI, like the Smartport Bus and AppleTalk, is mostly theoretical at the moment. While SCSI was designed to handle a wide variety of devices, the only thing you can hook to an Apple II through a SCSI card *right now* is a disk drive (and only a hard one at that).

Disk drives and the Smartport Bus. Both SCSI and Apple's Smartport are "intelligent" buses. This means that the devices connected to the bus are all supposed to have their own little computers built into them and are supposed to be able to respond to a few high-level commands. They don't have to be told in detail such stuff as how many stepper motor steps are needed to move their disk drive arm from track 12 to track 16. Dumb devices, on the other hand, have to be told everything in exhaustive electronic detail.

One major advantage the Smartport Bus has over SCSI, however, is that, in specific situations, it can also support dumb devices. The dumb device you'll most frequently find connected to the Smartport Bus is a 5.25 inch floppy disk drive. On the Apple IIc, you can plug one dumb 5.25 disk drive into the *end* of the Smartport daisy chain. It will appear to be connected to slot 6 as drive 2 (even though other Smartport devices in the chain will appear to be in slot 5). On the Apple IIgs, you can plug two 5.25 drives in, again at the *end of the chain*. They will both appear to be connected to slot 6.

Any Apple-compatible 5.25 drive is capable of working in a Smartport chain, incidentally, but not all of them have the right connector. Older drives have what's called a 20-pin header connector. Newer drives have the Smartport-compatible DB-19 connector. You can buy adapters to convert one to the other, or you can build an adapter using the instructions published here in December 1985 (page 90).



"I HAVEN'T LOCATED THE PROBLEM YET."

On the new Apple IIgs, you can also plug "Apple 3.5" dumb drives (as opposed to "UniDisk 3.5" intelligent drives) into the Smartport Bus connector. Apple 3.5 dumb drives work with the Smartport Bus only on the IIgs, however. The IIc Smartport Bus (available on any IIc that has the "3.5 ROM" or the "Apple Memory Card" upgrade) requires UniDisk 3.5 drives. And the IIe/II-Plus Smartport Bus (that is, the UniDisk 3.5 controller card), supports only UniDisk 3.5 drives. It was not designed to work with either 5.25 floppy drives or the non-intelligent Apple 3.5 drives.

Somewhere there is someone who knows for sure how many drives can be connected to the IIgs Smartport Bus connector, but we haven't found that wizard yet. All references agree that two 5.25 drives can be installed at the *end* of the chain and that these will appear to be connected to slot 6. All references agree that one or two dumb 3.5 drives can be connected at the *beginning* of the IIgs Smartport chain, and that these will appear to be connected to slot 5.

According to *Setting Up Your Apple IIgs,* "You can attach up to four drives to the port by connecting them to each other in a daisy-chain fashion....Only two of the drives can be 3.5 inch drives." (And only two can be 5.25 drives, we might add.)

According to the beta draft of the *Apple IIgs Firmware Reference Manual* (available from the Apple Programmers and Developers Association, see September 1986, page 2.57), "Current implementations of ProDOS only support two devices per port or slot. If more than two devices are connected to the device chain, devices beyond the second cannot be accessed with ProDOS 1.1.1. ProDOS 1.2 will support up to four devices on Smartport. ProDOS 1.2 will map the two devices beyond the second so that they will appear to be connected to slot 2....No remapping of units to slot 2 will be necessary with ProDOS 16 since ProDOS 16 will support more than two devices per port or slot.

And, from the beta draft of the *Apple IIgs ProDOS 16 Reference,* "Up to four devices may be connected to Smartport. However, because ProDOS 16 supports only 2 devices per slot, the third and fourth devices are treated as if they were in slot 2."

More about this shortly.

Distinguishing the bus from the bus station. As we've discussed in previous issues, the word "bus" is used in electronics circles to mean a group of wires or "lines" carrying related signals. It's ok, however, for those of you who aren't magnetically attracted to this meaning of "bus" to think of the Smartport Bus as a large yellow vehicle for moving data back and forth between your computer and the devices that are attached to it.

What *is* important, however, is that you distinguish between the Smartport *Bus,* which consists of cables, connectors, and electronic signals, and a Smartport-compatible *protocol converter,* which is the software interface that sits between your programs and the Smartport Bus. The reason it's essential that you make this distinction is that a Smartport-compatible protocol converter can also be used to access devices that are connected to other kinds of buses, or even to no bus at all.

Think of a Smartport-compatible *protocol converter* as a bus station. It is clearly marked so that all programs can find it. It has a consistent ticket window so that all programs can use it. What's beyond the bus station's departure gate, however, can be just about anything.

Apple's SCSI card, for example, uses a protocol converter that looks just like Smartport to access devices on the SCSI Bus. Most programs don't distinguish, nor should they, between a SCSI Bus device, a Smartport Bus device, or even an Apple Memory Card. All of these devices use a protocol converter that looks like Smartport, but only one of the three actually uses a Smartport Bus.

A Smartport-compatible *protocol converter* can support up to 127 devices. Each device on a single Smartport chain is assigned a *unit number.* The protocol converter itself gets to be unit number zero, the others are 1 through 127.

However, the Smartport *Bus* almost certainly doesn't have the spark to access that many devices. Its electronic signals will decay long before they get to the last device in the daisy chain. Likewise, a SCSI bus, by definition, will support nowhere near 127 devices.

Worse yet, as we saw a few paragraphs ago, ProDOS 1.1.1 doesn't want to deal with more than two devices per Smartport; developing versions of ProDOS appear to deal with only four devices per Smartport (though that's not totally clear). This could be a significant limit in terms of the IIgs, because in addition to any disk drives plugged into the slot 5 Smartport Bus connector, the IIgs supports both an internal RAMdisk and an internal ROMdisk.

These two devices also appear to be connected to the slot 5 Smartport daisy chain. (The IIgs automatically creates the RAMdisk, when instructed to through the control panel, out of extra RAM installed in the IIgs memory expansion slot. This memory expansion slot is also capable of holding ROM —although no cards are currently available that support this. This ROM can be configured several ways, including as a ROMdisk.) Thus, if the RAMdisk and ROMdisk were being used, the maximum number of devices (not counting one or two 5.25 dumb drives) that could be connected to a IIgs slot 5 Smartport Bus connector would be two.

How to find the bus station. If you are writing a program that needs to be curious about what kinds of storage devices are available to it, you can easily do a "slot scan" to look for Smartport-compatible protocol converters.

Slot scans have a long history. The tradition began with the Autostart Monitor in the Apple II-Plus. When you turn an Apple II-Plus or later Apple II on, the Monitor scans the slots, beginning with slot 7 and working downward, looking for a disk drive. It boots the first one it finds.

As we've explained in previous issues, each Apple II slot has a 256-byte memory space dedicated to it. What appears in this memory space is a set of machine language routines. These routines actually live in a ROM chip on the controller card plugged into that slot. (On the IIc and IIgs there are a number of built-in cards that follow the same format.) This address range, in hexadecimal, starts at $Cn00, where n is the slot number. In decimal this translates to byte 49152 + (SLOT*256).

If you let ADR = 49152 + (SLOT*256), then what the Autostart Monitor looks for is a 32 at ADR+1 ($20 at $Cn01), 0 at ADR+3 ($00 at $Cn03), 3 at ADR+5 ($03 at $Cn05), and 60 at ADR+7 ($3C at $Cn07). All 5.25 Disk II-compatible floppy disk controller cards have these values in these bytes.

In enhanced Apple IIes and later Apples, the Monitor doesn't look at all four of these identification bytes—just the first three. This is because Smartports and some hard disk controller cards don't have a 60 at ADR+7. It would take several paragraphs to explain why—and we did so just a couple of months ago—so if you'd like to know more see "How many drives have we here?" in the November 1986 issue, page 2.79. (And put a note on that article to refer to this one for more information about counting the number of drives connected to a Smartport protocol converter.) The important thing to note in this discussion is that since Smartport protocol converters don't have a $3C in that fourth ID byte, they won't automatically boot on startup when installed in older Apples.

What Smartport protocol converters do have in that fourth byte is a zero. Thus, the complete "signature" of a Smartport-compatible protocol converter is:

```
$Cn01 = $20      49153 + (SLOT*256) = 32
$Cn03 = $00      49155 + (SLOT*256) =  0
$Cn05 = $03      49157 + (SLOT*256) =  3
$Cn07 = $00      49159 + (SLOT*256) =  0
```

You can do a slot scan that identifies Smartports like this:

```
100 REM * Slot scanner *
110 FOR SLOT = 1 TO 7
120 : ADR=49152 + (SLOT*256)
130 : IF PEEK(ADR+1) < > 32 THEN 190
140 : IF PEEK(ADR+3) < >  0 THEN 190
150 : IF PEEK(ADR+5) < >  3 THEN 190
160 : IF PEEK(ADR+7) < >  0 THEN 190
170 : SMART(0)=SMART(0)+1 : REM count Smartports found
180 : PRINT "Smartport found in slot ";SLOT
190 NEXT : PRINT
200 IF SMART(0)=0 THEN PRINT "No Smartports found." : END
```

ProDOS and Smartport. Like the Monitor, ProDOS also does a slot scan. This happens as ProDOS is starting up. During this scan ProDOS attempts to identify all of the storage devices available on the computer and decides how it will deal with them. Like newer Monitors, it looks at just the first three of the four ID bytes.

If it finds a card that matches, it then looks at ADR+255. If this byte holds a zero, ProDOS assumes the card is a 16-sector floppy disk controller (and prepares to deal with it using its built-in "floppy driver" software). A 255 in this byte would indicate a 13-sector floppy controller, if such a thing still existed (in this case ProDOS would ignore the device). Any other value indicates that routines ProDOS should use to deal with the device are stored on the card itself (in this case ProDOS assumes the entry point to these routines is at ADR + whatever was in ADR+255).

ProDOS was specifically designed to deal with disks and other storage devices. In general, it's best to leave things such as buying Smartport Bus tickets and putting blocks of data on the bus to ProDOS. It's much easier to type in CATALOG and get a list of the files on the disk in the drive connected to the Smartport Bus than it is obtain the same information by driving the bus yourself.

Even so, ProDOS doesn't know a Smartport from a Dumbport. What today's ProDOS sees when it runs into a Smartport protocol converter is a ProDOS-compatible block storage device. ProDOS expects such devices to respond to just four commands—status, read, write, and format. In order to execute one of these commands, ProDOS stores the command number and other needed data in bytes $42-$47 (66-71) and calls the card's ProDOS entry point.

You and Smartport. By going directly to Smartport yourself, however, there are a few things you can accomplish that ProDOS can't. For example, ProDOS has no command for ejecting a disk from a 3.5 drive. Smartport does.

Smartport protocol converters have a number of commands. Most commands can be sent to any device in the daisy chain. Most of the commands are useless for most programmers. For example:

Init. This command resets the Smartport and all devices connected to it. This command can't be made to a specific device, but only to the Smartport as a whole. Leave this command alone.

Format. This call tells a disk drive, or any "block storage" device, to erase itself and prepare all blocks on the medium for reading and writing. Operating-system specific information, such as free block or free sector maps or blank directories are not a part of this call. A device that requires additional information to format itself is supposed to respond to this call by indicating that a successful format was completed without actually doing anything. I recommend you leave this command alone. Use the format command in the ProDOS Machine Language Interface if you really want to destroy...I mean format...something.

ReadBlock, WriteBlock. These commands read or write a block of data on the storage device. They are essentially duplicates of the ProDOS read block and write block commands. ProDOS-based applications are better off using the ProDOS MLI read block and write block commands than these.

Open, Read, Write, and Close. These are special Smartport commands for character-oriented (as opposed to block-oriented) devices, such as printers and modems. Since there aren't any devices like this currently available, there isn't much you can do with these commands. One thing somebody could do, but hasn't, is read and write 3.5 inch Macintosh disks. These commands must be used for this because Macintosh disks have 524-byte blocks, rather than the ProDOS standard 512-byte blocks that ReadBlock and WriteBlock use. There is no technical reason why programs such as Apple's *System Utilities* or *Copy II Plus* couldn't copy files to and from Macintosh disks as easily as they do to and from DOS 3.3 disks. The software just hasn't been written yet. Interested parties can obtain more information from Apple's "UniDisk 3.5 Technote #4," pages 4 to 7.

There are two Smartport commands that are kind of fun. They are:

Status. This command can be sent to the Smartport itself or to any device connected to it. It comes in four flavors that return various types of information about the Smartport or about the device connected to it.

Control. This command is used to send special control commands to devices. The 3.5 disk eject command is a special version of this call.

A typical Smartport protocol converter call looks like this:

```
JSR DISPATCH      the DISPATCH address is $Cn00 + what's in $CnFF + 3
.DA #CMD          one-byte command number
.DA PARMLIST      two-byte parameter list pointer
BCS ERROR
```

Notice that the command number and parameter list pointer are embedded right in the assembly language code. This is similar to how a ProDOS MLI call is done. The Smartport will jump over these bytes when returning control to the main program. Thus, the next statement executed will be the Branch on Carry Set. If an error has occurred, the microprocessor's carry bit will be set and the accumulator will hold the error number. If, on the other hand, there was no error, the carry bit will be clear and the accumulator will hold a zero.

The Smartport entry point is always three bytes beyond the ProDOS entry point. We can have our slot scan routine from a few paragraphs ago find this this address, which Apple's documentation calls DISPATCH, by adding the following line:

```
173 : SMART(SLOT)=ADR + PEEK(ADR+255) + 3 : REM get DISPATCH address
```

There are two types of Smartport calls, standard and extended. All Smartport protocol converters support standard calls. At the moment, the only one that supports extended calls is the one built into the IIgs. After a quiet paragraph or two for those who care, we'll ignore extended calls.

(Extended calls have four-byte memory and block pointers. Standard calls have two-byte memory and three-byte block pointers. Extended calls are useful primarily on the IIgs (and future machines) because they allow data to be saved from or loaded into any of the many banks of linear memory.

(Interestingly, a four-byte block pointer, combined with standard ProDOS 512-byte blocks, allows a maximum volume size of over 2 trillion bytes. This is the equivalent of about 512K for every Apple II ever manufactured. It should be enough to last for quite some time.

(You can tell if a Smartport supports extended commands by looking at byte $CnFB (ADR+251). If the high (seventh) bit of this byte is 1, then extended commands are supported. In addition, bit 1, when set to 1, indicates the Smartport is connected to a SCSI bus. Bit 0, when set to 1, indicates the Smartport controls a RAMcard. Thus:

```
175 B=PEEK(ADR+251)
176 IF B=>128 THEN B=B-128 : PRINT "Extended ";
177 IF B=2 OR B=3 THEN PRINT "SCSI ";
178 IF B=1 OR B=3 THEN PRINT "RAMcard ";
```

From Applesoft, we can make a Smartport call like this:

```
1100 REM *  CALL SMARTPORT  *

1110 POKE 768,32  : REM JSR DISPATCH
1120 POKE 770,SMART(SLOT)/256
1130 POKE 769,SMART(SLOT)-PEEK(770)*256
1140 POKE 771,CMD
1150 POKE 772,11  : REM parmlist address is $30B (779)
1160 POKE 773,3
1210 POKE 774,141 : REM STA 778
1220 POKE 775,10  : REM error code address is $30A (778)
1230 POKE 776,3
1240 POKE 777,96  : REM RTS
1250 CALL 768
1260 ERR=PEEK(778)
1270 RETURN
```

After making a call to a Smartport protocol converter, we should always check to see whether an error occurred. This Applesoft Smartport caller stores the returned error code in byte 778. If this code is a zero, no error occurred. If it's not zero, the following lines tell us what happened:

```
1900 REM * Smartport error messages *

1901 IF ERR= 1 THEN PRINT "$01. No such command."
1904 IF ERR= 4 THEN PRINT "$04. Bad parameter list."
1906 IF ERR= 6 THEN PRINT "$06. Smartport Bus error."
1917 IF ERR=17 THEN PRINT "$11. This device can't respond to this command."
1931 IF ERR=31 THEN PRINT "$1F. Interrupts not supported by this Smartport."
1933 IF ERR=33 THEN PRINT "$21. This device can't respond to this subcommand."
1934 IF ERR=34 THEN PRINT "$22. Bad Control List."
1939 IF ERR=39 THEN PRINT "$27. I/O error."
1940 IF ERR=40 THEN PRINT "$28. No such unit number."
1943 IF ERR=43 THEN PRINT "$2B. Disk is write protected."
1945 IF ERR=45 THEN PRINT "$2D. No such block number on device."
1947 IF ERR=47 THEN PRINT "$2F. No disk in drive/device is offline."
1948 IF ERR>47 AND ERR<64 THEN PRINT "$30-3F. Device-specific fatal error."
1964 IF ERR>63 AND ERR<80 THEN PRINT "$40-4F. The future has arrived."
1980 IF ERR>79 THEN PRINT "$50-5F. Device-specific non-fatal error."
1999 RETURN
```

Before using the Applesoft Smartport caller we have to set SLOT equal to the slot of the Smartport we want to call, CMD to the command number we want (0 for Status, 4 for Control), and we have to fill out a parameter list. The parameter lists for the Status and Control commands are exactly the same. They start with a one-byte parameter count (which should be 3 for both commands), then have a one-byte unit number (zero for the Smartport itself, 1 to 127 for devices), then have a two-byte pointer to a Status List or a Control List, and finally have a one-byte subcommand. From Applesoft, we can set this up with:

```
1000 REM * Set up parameter list *
1010 POKE 779,3
1020 POKE 780,UNIT
1030 POKE 781,16 : REM Status/Control List address is $310 (784)
1040 POKE 782,3
1050 POKE 783,SUBCMD
```

What buses stop here? To find out how many devices are connected to a Smartport, you do a status call with a subcommand of zero. The number of devices will be returned in the first byte of the Status-List. By adding the following commands to all the stuff that's gone so far, you have a program that scans a computer's slots looking for Smartports and then tells you how many devices are connected to each one found:

```
300 UNIT=0   : REM call goes to Smartport itself
310 CMD=0    : REM Status call
320 SUBCMD=0 : REM we're asking for the number of units
330 FOR SLOT=1 TO 7
340 : IF SMART(SLOT)=0 THEN 390
350 : GOSUB 1000
360 : IF ERR THEN PRINT "ERROR--"; : GOSUB 1900 : GOTO 390
370 : NDEV(SLOT)=PEEK(784)
380 : PRINT "The Smartport in slot ";SLOT;" has ";NDEV(SLOT);" devices."
390 NEXT : PRINT

999 END
```

(An additional bit of information you can pick up from this call comes back in the second byte of the Status List, byte 785 in this case. If bit six of this byte is clear, it means that a device on the Smartport chain is sending an interrupt. This can only happen on the IIc and will only happen after someone invents a Smartport Bus device that sends interrupts.)

If you're like most programmers, you'd like a bit more information about the devices connected to the bus than simply how many there are. A Status call to a *device* (rather than to the Smartport itself) with a subcommand of 0 will return four bytes. The bits of the first byte, which is called the Device Status byte, give various kinds of information that I'll show you how to decipher in a moment. After that come three bytes that indicate the number of blocks on the device.

I can see absolutely no reason to ever use subcommand 0, however, because a Status call to a device with a subcommand of 3 returns a Device Information Block or DIB. This 25-byte block starts off with the same four bytes that subcommand 0 sends. In addition, after that you get a word, such as "RAMDISK" or "DISK 3.5," that describes the device (the length of the word is in the fifth byte, the device name itself in the next 16 bytes, padded with spaces at the end if necessary). After that there are device type and subtype bytes and a two-byte version number.

From Applesoft it is always a mess to figure out which bits in a byte are ones and which are zeros. We'll have to do it here, however, so I have a magic subroutine for you. You tuck the byte you want to analyze into D, call the subroutine, and it sets the values B(7), B(6),... B(0) to one or zero depending on the value of the equivalent bit:

```
1500 REM * Turn a byte's bits into an array *
1501 REM *       D=byte to analyze           *
1502 REM *   B(7)=array with bit values      *
1510 FOR N=7 TO 0 STEP -1
1520 : BN=2^N : B(N)=0 : IF D>BN-1 THEN D=D-BN : B(N)=1
1530 NEXT
1540 RETURN
```

Now we have everything we need to find out everything that can be known about any device attached to a Smartport protocol converter. There's a lot of information here. Try this:

```
400 FOR SLOT=1 TO 7
410 : IF SMART(SLOT)=0 THEN 699
420 : FOR UNIT=1 TO NDEV(SLOT)

430 : : CMD=0 : SUBCMD=3
440 : : GOSUB 1000
450 : : IF ERR THEN PRINT "ERROR--"; : GOSUB 1900 : GOTO 698

460 : : DEVST=PEEK(784)
470 : : NBLKS=PEEK(785) + PEEK(786)*256 + PEEK(787)*65536

480 : : NAME$="" : FOR I=1 TO PEEK(788) : NAME$=NAME$+CHR$(PEEK(788+I)) : NEXT
490 : : TYPE=PEEK(805) : SBTYPE=PEEK(806)
500 : : VERS=PEEK(807) + PEEK(808)*256

510 : : PRINT "Device ";UNIT;" in slot ";SLOT;" is a ";NAME$
520 : : PRINT "  It has ";NBLKS;" blocks of storage capacity."
530 : : D=DEVST : GOSUB 1500

540 : : IF B(7)=0 THEN PRINT "  It is a character-oriented device."
541 : : IF B(7)=1 THEN PRINT "  It is a block-oriented device."

542 : : IF B(6)=0 THEN PRINT "  You can't write to this device."
543 : : IF B(6)=1 THEN PRINT "  Writes to this device are allowed."

544 : : IF B(5)=0 THEN PRINT "  You can't read this device."
545 : : IF B(5)=1 THEN PRINT "  This device is readable."

546 : : IF B(4)=0 THEN PRINT "  This device is offline (disk ejected)."
547 : : IF B(4)=1 THEN PRINT "  This device is online (disk inserted)."

548 : : IF B(3)=0 THEN PRINT "  You can't format this device."
549 : : IF B(3)=1 THEN PRINT "  Formatting this device is allowed."

550 : : IF B(2)=0 THEN PRINT "  The device isn't write protected."
551 : : IF B(2)=1 THEN PRINT "  The disk is write protected."

552 : : IF B(1)=0 THEN PRINT "  This device is not interrupting."
553 : : IF B(1)=1 THEN PRINT "  This device is the interrupt culprit."

554 : : IF B(7)=1 THEN 560 : REM lines 555-6 are for character devices only
555 : : IF B(0)=0 THEN PRINT "  This device is closed (available)."
556 : : IF B(0)=1 THEN PRINT "  This device is open (busy)."

560 : : PRINT "  The TYPE byte of this device indicates it is a ";
570 : : IF TYPE=0 THEN PRINT "RAMdisk."
571 : : IF TYPE=1 AND SBTYPE=  0 THEN PRINT "UniDisk 3.5."
572 : : IF TYPE=1 AND SBTYPE=192 THEN PRINT "Apple 3.5."
573 : : IF TYPE=2 THEN PRINT "hard disk."
574 : : IF TYPE=3 THEN PRINT "SCSI device."

580 : : IF TYPE < 2 THEN 600 : REM see text about SUBTYPE byte
581 : : PRINT "  The SUBTYPE byte of this device indicates that it: "
585 : : D=SBTYPE : GOSUB 1500

590 : : IF B(7)=1 THEN PRINT "    Supports extended Smartport calls."
591 : : IF B(6)=1 THEN PRINT "    Supports disk-switched errors."
592 : : IF B(5)=1 THEN PRINT "    Supports removable media."

600 : : PRINT "  The version number of this device is ";VERS;"."

698 NEXT : PRINT
699 NEXT
```

More Smartport commands. In addition to Status subcommands 0 and 3, subcommands 1 and 2 have also been defined for the Smartport Status command. Subcommand 1 tells a device to return its "device control block" or DCB. Since every Smartport device has its own unique format for this block it is difficult to say much about this subcommand without going into painful detail. Subcommand 2 is designed to allow character-oriented devices to return their "newline" status. More on this when there is such a thing as a Smartport character-oriented device.

The Smartport Control command has four device-independent subcommands. They are all pretty useless for most of us. Subcommand 0 resets the device to its default values. Subcommand 1 sends the device a new Device Control Block. Subcommand 2 sets the newline status. Subcommand 3 is for servicing an interrupt.

Of somewhat more interest are the device-*dependent* Control commands. The non-intelligent Apple 3.5 disk, for example, has seven of these, implemented as subcommands 4 though 10. The most useful is subcommand 4, which ejects disks from the drive. The others are mostly used for copy protection and deprotection, both of which are a waste of human resources.

The UniDisk 3.5 implements four Smartport Control subcommands and one Status subcommand. Again, Control subcommand 4 ejects disks from the drive. Subcommands 5, 6, and 7 allow you to download programs into the drive and execute them. Again, this is mainly useful for copy protection and deprotection, however, it may be possible for some wizard to do something as magic as getting one UniDisk to automatically back up another without the host Apple II knowing anything about it. Notice I said "may." The UniDisk 3.5 Status subcommand 5 allows an application to get more detail about read/write errors, including the contents of the registers of the UniDisk's 6502 microprocessor.

Just for fun, let's add a few lines to our program that will eject the disk from any Apple 3.5 or UniDisk 3.5 drive that is found:

```
610 : : IF TYPE < > 1 THEN 698 : REM all 3.5 drives (so far) have TYPE=1
620 : : CMD=4 : SUBCMD=4
630 : : POKE 784,0 : POKE 785,0 : REM # of items in Control List
640 : : GOSUB 1000
650 : : IF ERR THEN PRINT "ERROR--"; : GOSUB 1900 : GOTO 698
```

Line 630 is quite important. Just as the Status command passes back a Status List, the Control command requires that you pass it a Control List. The first two bytes of the Control List are supposed to indicate the number of items the Control List has. In the case of Eject, there are none. So you must put zeros in the first two bytes of the Control List or strange things happen.

SYS.DESKTOP. It takes possession of all available memory. Tom Vanderpool (ed note: a local wizard) and I figured out that you can save a dummy file of whatever size you want in the partition, then bring up AppleWorks. The desktop takes up the rest of the partition. From inside AppleWorks, you can delete the dummy file and use its memory for other files. Another possibility is to create a special partition just for SYS.DESKTOP, as it will not overwrite other partitions.

"CirTech's Flipper (called 'Flipster' in the U.S.) supports ProDOS, Pascal 1.3, and up to two 400K DOS 3.3 volumes. CirTech's version of CP/M Plus will automatically recognize and use the card. Supplied software allows up to four equal-sized partitions for support of Pascal 1.1, 1.2, and 1.3, ProDOS, DOS 3.3, and Microsoft Softcard II CP/M versions 2.20 and 2.23, and can create bootable RAMdisks for DOS, CP/M, ProDOS, and Pascal. The partition manager loads into the Flipper itself. The partition sizes are not as flexible as those of RamFactor. Flipper version 1.03-1.09 ROMs require a software patch to AppleWorks 1.3 to expand the desktop automatically as it is booted; version 2.00 eliminates this restriction. The Flipper software doesn't expand the database or word processor, neither does it allow AppleWorks to run on a II-Plus.

"Apple's own Apple Memory Card includes RAMdisk support for ProDOS, Pascal 1.3, and DOS 3.3 (DOS is limited to a maximum of a single 400K volume). AppleWorks 1.3 will recognize the card and use its RAM for an expanded desktop but does not expand the database or word processor as does RamFactor. AST has a card called Sprintdisk that is similar to Apple's Memory Card.

"Of the cards looked at here, RamFactor has the edge for users whose main concern is AppleWorks support; Flipper has the edge for those whose main concern is support of Apple Pascal or CP/M."

IIgs questions and observations

I recently received an Apple IIgs, AppleColor Monitor, and 3.5 drive. It replaces an enhanced IIe, but not without some surprises. Having owned every variety of Apple II computer to come out of Cupertino, I thought I could handle any problem, but your help would be appreciated.

The IIgs does not always act like a Super Serial Card. When I try to LIST an Applesoft program with PR#1 and LIST, the printer does not perform a carriage return until the end of the Applesoft line; that is, if the line is longer than the width of the carriage, it simply prints the rest of the line over the first part, from right to left. Is there a command to correct this (e.g. PRINT CHR$(9);"80N")? For now, I use the control panel to turn on echoing, which slows down the printing but corrects the problem.

The clock/calendar works differently from the previous Thunderclock standard. My files under ProDOS 1.1.1 are not being time/date stamped; instead <NO DATE> is printed in the catalog.

My ancient Disk II drives don't work with the IIgs, though third-party 5.25 drives work fine. The 3.5 works beautifully, except you have to remember to eject the disk before you turn off the machine. The 800K disks are great for backing up Sider volumes, but the Sider is still faster.

My Sider works just as well on the IIgs, but the connector is too large for the IIgs's back panel hole. I called First Class Peripherals, who told me the problem is being worked on. In the meantime, they said, I should simply feed the cable out the hole and let the plug dangle. Other than this, however, the Sider works flawlessly with the IIgs.

MultiScribe (version 1.1) does not work. I read a IIgs version is due out soon.

The color monitor is beautiful, although I get color fringes on some programs and other color programs don't always look good on the monitor. It reminds me of Apple RGB Monitor problems. The Control Panel is great for controlling the monitor display and adjusting the bell's sound. The keyboard is comfortable, but its snap-together construction is a bit flimsy. Also, the cables connecting the keyboard to the computer on one side and the mouse on the other detracts from its detached nature.

Richard Katz
Los Angeles, Calif.

My Super Serial Card acts just like your description of the IIgs. To get it to stop overprinting you have to do a PRINT CHR$(9);"80N" to set a line width and a PRINT CHR$(9);"C" to get it to start sending a Return when the designated line width has been reached. This is, incidentally, different from the IIc, which starts up with both those commands in effect.

The IIgs clock does not appear "in a slot," thus it can't be accessed in the same manner as the Thunderclock that ProDOS 1.1.1 expects. ProDOS 1.2 contains the driver necessary to read the IIgs internal clock. You should have gotten a IIgs System Disk with your machine, although it wasn't in the box with the first units shipped. Complain—the place you got your machine from can get you a copy. It has ProDOS 1.2 on it along with other goodies. See the next letter for how to get it onto your applications disks.

We know of people using Disk II drives on a IIgs with the Disk II controller card installed in slot 6 (remember to go to the Control Panel and select "your card" for slot 6). We know of no theoretical reason that would prevent Disk II drives, with the proper cable adapter, from working when plugged directly into the end of the IIgs Smartport chain, but we haven't actually seen that done.

You don't really have to eject the 3.5 disk before turning the machine off if you're willing to store the disk inside the drive until the next time you turn the computer on. (The problem is that you can't get it out of the drive with the power turned off.) This is how Apple's 3.5 drives work no matter what machine they're connected to.

***Multiscribe** isn't the only word processor that doesn't work on the IIgs. Incredibly, Apple's own **Apple Writer** has problems. The ProDOS version won't print through the IIgs serial ports, although it will print correctly through a serial card installed in a IIgs slot. The earlier "IIe" DOS 3.3 version won't automatically load the PRT.SYS and TAB.SYS files. The "2.0" DOS 3.3 version won't boot. No updates have been announced.*

More IIgs observations

The difference in cost between the IIgs with mouse and system disk ($999) and the IIgs upgrade ($499 + mouse + system disk) was not worth giving up my IIe. Apple's $250 rebate on an Imagewriter II made investing in a second system even more reasonable.

Although the *IIgs Owner's Guide* said my dealer could upgrade ProDOS on my application disks to version 1.2, my dealer didn't know the procedure. What you have to do is copy the file /SYSTEM.DISK/SYSTEM/P8 to the main directory of your application disk, then delete the old PRODOS file and change the name of P8 to PRODOS. Any copy utility could make the transfer, but I found the *copy a file* option on the IIgs Desktop program easy to use. Updating to ProDOS 1.2 lets me access the Control Panel, which is helpful especially when I need to change the RGB display from color to monochrome or back. (The *Catalyst 3.0* Desktop is really blurred when the IIgs is configured for color, so being able to interrupt Catalyst to change the display to monochrome is great. Then, when I use *Catalyst* to choose a program such as *Beagle Graphics*, I can change the display back to color.)

Apple Pascal 1.3-based programs can be a nuisance on the IIgs. MECA's *Managing Your Money*, for example, recognizes memory storage devices (disks or cards) only in slots 4, 5, and 6, and works best with two 3.5 drives and a memory expansion card. If I configure part of the IIgs memory as a RAMdisk, the IIgs makes the RAMdisk appear to be in slot 5, drive 2, and moves my second 3.5 drive to slot 2, where Apple Pascal can't find it. To let both 3.5 drives appear in slot 5, I have to set the IIgs RAMdisk size to zero and instead use my IIe's Apple Memory Card in slot 4 or slot 6 of the IIgs. But putting it in slot 4 disables the desktop mouse, while putting it in slot 6 disables my 5.25 drives.

Catalyst 3.0 will recognize the IIgs RAMdisk as being on line, but will not automatically load applications into it. However, it will load applications into an Apple Memory Card in any slot.

Some applications recognize the old Apple mouse, but not the new Apple desktop mouse. I have to put my old Apple mouse in slot 4, and configure the Control Panel for "your card" in slot 4, for applications such as Electronic Arts' *Music Construction Set*. (Soon *Deluxe Music Construction Set* should be out and I can put my Apple mouse and Mockingboard back in my IIe.)

AppleWorks 1.3 can access the IIgs RAMdisk, but the desktop doesn't expand into it as it does into the Apple Memory Card RAMdisk.

The price of the Apple IIgs Memory Expansion Card with 256K ($129) is lower than most of the third-party cards. I saved even more by ordering more sets of 256K chips from other sources and installing them myself. Sony speakers designed for the Walkman work fine with the IIgs and are less than half the price of the Bose Roommate speakers that Apple recommends.

Sam L. Pemberton, Jr.
Gallup, N.M.

As you point out, the two main problems with running ProDOS 1.1.1 on a IIgs are that the files won't be date stamped, because ProDOS 1.1.1 can't read the IIgs clock, and that ProDOS 1.1.1 disables interrupts, which means you can't get to the Control Panel with the usual three-key control/open-apple/escape command. ProDOS 1.2 fixes both of these problems. (Another route to the Control Panel, surer but more disruptive, is control/option/reset.)

*The file called PRODOS in the main, or root, directory of the IIgs System Disk will confuse lots of people. It's not really ProDOS at all, but a little program that runs when the disk is started up. It, in turn, runs P16, which is in the subdirectory called SYSTEM and is the real ProDOS 16. ProDOS 16 looks for a file called START in the SYSTEM subdirectory. If it can't find START, it runs the first **root directory** program with a name ending in ".SYS16". However, if there is a system program with a name ending in ".SYSTEM" before the first SYS16 program, then ProDOS 16 gives up and runs SYSTEM/P8, which is*

Ask (or tell) Uncle DOS

Last month I indicated that I couldn't boot from my Apple Memory Card (page 2.87). This month I discovered that my problem was that I didn't read the manual. The card has to be initialized with a program such as FILER or ***Copy II Plus*** *to make it a bootable device. This step may seem unnecessary, since the card will automatically format itself for ProDOS the first time you access it. The problem with auto-format is that it doesn't put some necessary "boot code" into the RAMdisk's block 0. The boot code loads and executes the file named PRODOS—without it nothing happens. Both FILER and* ***Copy II Plus*** *will install the necessary boot code in block 0.*

The Apple IIgs has created lots of mail and interest, although hardly anyone has actually been able to buy one. Robert Irwin in Federal Way, Washington writes that he paid for a IIgs exactly one-half hour after dealers were allowed to display the machine on September 25, yet as of December 2 his dealer had not been able to deliver. From Ann Arbor, Michigan, Clyde Godfrey writes that his dealer is now quoting January/March delivery for the machine he ordered September 27. A few subscribers have gotten their hands on the machine, however, and this month we'll print a few of their letters and answer a few IIgs questions. But first, a word from our conscience:

Auxmem corrections and more

I'll bet the author of your auxiliary memory tester (December 1986, page 2.83) writes programs while using 80-column mode. If you run the program in 40-column mode it doesn't work. I suggest you add a line to the program to turn on 80-columns.

Pete Ross
Wayne, Mich.

Oh, embarrassment! Here I am, always insisting that programmers put everything back where they found it, and I left STORE80 and PAGE2 dangling. As you point out, this is no problem in 80-column mode, but in 40-column mode the program hangs when it tries to execute the PRINT in line 400.

Worse yet, in trying to find the source of the bug I found yet another bug I added at the last minute last month. I was concerned that my original version of the program (which worked fine) would mess up any data stored in a RAMdisk in auxiliary memory, so I added some lines to save and restore the contents of the auxiliary memory bytes that the program overwrites. Because of the cheap way I did this, the program shows some auxiliary-slot cards have four times more memory than they really have.

Here are some changes to the program that fix both bugs:

```
150 S8=PEEK(49176) : POKE 49153,0
160 P2=PEEK(49180) : POKE 49237,0

DELETE 230
250 FOR BANK=127 TO 0 STEP-1
260 POKE 49267,BANK : REM $C073
270 POKE 8192,BANK : REM put bank number at $2000
280 NEXT                                     49512

381 IF P2<128 THEN POKE 49236,0 : REM fix PAGE1
382 IF S8<128 THEN POKE ~~49238~~,0 : REM fix STORE40
```

Lines 150 and 160 now save the status of the PAGE1/PAGE2 and STORE40/STORE80 switches before messing with them; lines 381 and 382 restore the switches to their original status. With these changes the program will work in either 40-column or 80-column mode. The additions and deletions in lines 230 to 280 get the program to stop having quadruplets.

Last month Dennis also worked up some notes for me on the differences between the "Apple II standard" memory cards that are currently available. I couldn't find the notes when I needed them, but I've found them since. This stuff is intrinsically interesting, so I thought I'd let everybody in on what Dennis discovered:

"Applied Engineering's RamFactor comes with software that expands the AppleWorks 1.3 desktop and allows a few more than 5,000 database records and 5,000 word processor lines. The card has built-in RAMdisk support for Pascal 1.3, DOS 3.3, and ProDOS. Recognition of the RAMdisk by ProDOS and Pascal is automatic; DOS 3.3 requires an IN#n command to enable the card. The firmware allows for the RAMdisk to be divided into as many as nine partitions, each of which can use any of the supported operating systems. Software on disk allows placing DOS 3.3 into a partition that can be booted from the partition manager. PR#n calls up the partition manager. A 1 meg RamFactor can be partitioned into two 400K DOS 3.3 drives. It connects into DOS 3.3 at a different location than the Apple Memory Card, so the two will operate together with DOS 3.3. Maximum memory on the card is 1 meg; it can be expanded past this but the cost is prohibitive at present. Supplied software allows running AppleWorks 1.3 on an Apple II-Plus equipped with a RamFactor and a Videx-compatible 80-column card. The expanded desktop appears in the current partition as a file of type $00 named

Known Bugs. The SUBTYPE byte returned in the Device Information Block is pretty strange. Apple's documentation says that a 1 in bit 7 of this byte indicates the device supports extended Smartport calls; a 1 in bit 6 indicates the device supports "disk switched errors," whatever they are; and a 1 in bit 5 indicates the device has non-removable media. However, few, if any, of the currently available devices actually follow this protocol. That's the reason for line 580. Also see lines 571 and 572.

Apple's documentation lists only specific devices in relation to the TYPE byte and doesn't claim there is any pattern; however, 0 appears to be reserved for RAMdisks, 1 for Smartport Bus 3.5 disks, 2 for Smartport Bus hard disks, and 3 for SCSI devices. Ask me again in ten years and we'll see how well I guessed this one.

We've known for some time that there is a bug in the ProDOS/UniDisk 3.5 relationship that causes bad things to happen when you write protect 3.5 inch disks (see January 1986, page 98 and February 1986, page 2.7). Here's the bug. Before doing a write to a disk, ProDOS makes a status call. It wants to make sure there is a disk in the drive and that the disk isn't write-protected. If the status comes back ok, ProDOS messes around a bit and then does the write.

Remember that there is a difference between a ProDOS status call and a Smartport status call. When ProDOS does a status call, it puts a bunch of stuff in bytes $42-47 and calls the ProDOS entry point. ProDOS expects the device to send back a $27 (39) if an I/O error occurs, a $28 (40) if no device is connected, or a $2B (43) if the disk is write protected. If no error occurs, it figures a write-enabled disk is in the drive and ready to go.

On the other hand, the Smartport Status command tells you whether a disk is write-protected (lines 550-551) but it doesn't return an *error* if it is. Only a WriteBlock call will do that. The original UniDisk 3.5 Smartport-to-ProDOS routines forgot to dig the write-protect situation out of the Device Status byte and send ProDOS an error if write-protect was on. Consequently, ProDOS didn't find out a disk was write-protected until it actually tried to write on the disk—and by then it was too late to prevent ProDOS from becoming quite confused.

This situation was fixed on the IIgs by adding a few bytes to the Smartport-to-ProDOS routines that turn the write-protected bit in the Device Status byte into a real error for ProDOS status calls. You can figure out whether your Smartport controller has this fix or not by adding the following lines to our program:

```
660 PRINT
661 PRINT "Please open the write-protect hole on this disk,"
662 INPUT "  then reinsert it. Press <RETURN> when ready.";A$
663 PRINT
670 POKE 66,0 : REM for ProDOS status command, put a zero at $42
671 S1=SLOT : IF UNIT>2 THEN S1=S1-4 : REM convert to ProDOS unit #
672 S1=S1*16 : IF UNIT=2 OR UNIT=4 THEN S1=S1+128
673 POKE 67,S1 : REM put ProDOS unit # at $43 (dsss0000)
675 POKE 809,32                               : REM JSR DRIVER
676 POKE 811,(SMART(SLOT)-3)/256
677 POKE 810,(SMART(SLOT)-3)-PEEK(811)*256
678 POKE 812,176 : POKE 813,2                 : REM BCS 816
679 POKE 814,169 : POKE 815,0                 : REM LDA #0
680 POKE 816,141 : POKE 817,53 : POKE 818,3 : REM STA 820
681 POKE 819,96                               : REM RTS

684 IF PEEK(820) < > 0 THEN 690
685 PRINT "This Smartport has the UniDisk 3.5 write-protect bug."
686 PRINT "  Eject this disk, cover the write-protect hole, and don't"
687 PRINT "  ever put a write-protected disk in this Smartport again."
688 PRINT : INPUT "Press <RETURN> to continue.";A$
689 GOTO 698

690 PRINT "  You can safely use write-protected disks on this Smartport."
```

Have I mentioned that Smartport calls require 30 to 35 bytes of stack space, can't be used to transfer data into or out of the zero page, and, on the IIgs, must be made from emulation (standard Apple II) mode rather than native mode?

Oh, and finally,... don't worry. I've left computers alone together for weeks at a time and not one of them has ever gotten pregnant.

really ProDOS 8. ProDOS 8 then runs the program with the name ending in SYSTEM. What this all means is that the technique you outline for moving P8 and renaming it PRODOS makes 8-bit applications boot much faster than they do if you run them from a ProDOS 16 system disk.

ProDOS 1.2 (which is the current version of "ProDOS 8" or "P8") comes complete with the track 0 bug discussed at length in our November issue (page 2.74)—Apple's developer technical support people have indicated to me that this bug will be fixed in the next releases of ProDOS 8 and ProDOS 16. To adapt November's program so that it works with ProDOS 1.2, copy P8 and rename it PRODOS. Then make the following changes:

```
200 FOR 22723 TO 22732 rest of line unchanged
300 ADR=20996 rest of line unchanged
550 PRINT "This version of ProDOS not 1.2."
```

There may be several release dates of ProDOS 1.2 floating around. The official version bears the date 06-SEP-86.

AppleWorks addendum

I think your suggestion that Lawrence Pratt (November 1986, page 2.77) change from *General Manager* to AppleWorks is a good one, but with one caution. Anyone who tries to use AppleWorks without added memory and without *AutoWorks* will never know the true pleasure and power of AppleWorks. Added memory allows AppleWorks to be completely loaded into memory and eliminates disk access. Those of us who use AppleWorks this way soon forget the awkwardness and slowness of the standard, disk-based AppleWorks.

Autoworks adds such power to AppleWorks as to make it almost a brand new program. In suggesting AppleWorks to new users, I am always careful to add the suggestion for memory and *AutoWorks* as well.

When I use *Copy II Plus* to rename AppleWorks files, I use all upper case (which is the only way *Copy II Plus* will allow). When the files are loaded into AppleWorks, however, the names are a random mix of upper and lower case letters. AppleWorks loads the files just fine and the files are all intact, but the first time one looks at the AppleWorks directory, it appears a disaster has struck.

By the way, I know a number of AppleWorks SIG's around the country are cataloging all the disks from TAWUG and setting up data bases with descriptions of the files in the disks. We are doing the same, and if some of the SIGs would like to share their work so that we don't all have to re-invent the wheel, please have them contact me.

Thomas E. Militello, M.D.
Chairman, The AppleWorks S.I.G.
Original Apple Corps
Los Angeles, Calif.
(213) 541-2766

ProDOS officially allows only upper case letters in filenames. For the three AppleWorks filetypes, however, the 16 bits in a pair of bytes that are part of every file's directory entry (the AUX_TYPE bytes) were defined as indicators of which characters of the file's name are upper case and which lower. In addition, "lowercase" periods are defined as space characters. This is a very creative use of AUX_TYPE and one of the many fine touches of AppleWorks.

When you catalog a disk of AppleWorks files with Basic.system, however, the filenames will appear in upper case, with periods instead of spaces, just like all other ProDOS filenames. This is because Basic.system doesn't know anything about AppleWork's special AUX_TYPE protocol. From inside AppleWorks, however, the names appear in upper and lower case and include spaces as well as periods.

*When you change a file's name with **Copy II Plus** or Basic.system, you change the letters in the name but not the AUX_TYPE byte. Consequently, you get the previous mix of upper and lower case laid over the top of your new name—and the strange results you encountered. Versions of **Copy II Plus** beginning with 7.1 will correctly handle the AUX_TYPE bytes on AppleWorks files.*

Another escape route

A sharp AppleWorks idea I read about a while back is to make a custom printer and set one of the printer options to just ESC. If you make the bold on and bold off codes "ESC", for example, you can do a control-B followed by your printer's ASCII command code and the printér will see it as a printer command. Inside AppleWorks your text might look something like:

```
The ^K2APPLE IIgs^K0 is a ^K3great^K0 computer!
```

When you print this you can get special options that Appleworks doesn't normally allow, such as color on the ImageWriter II printer.

Mike Mitchelson
Overland Park, Kans.

Single sheet sticky space

When I print on single-sheet letterhead from AppleWorks, my printer beeps and locks everything up because it "runs out of paper" while advancing the single sheet out of my printer. I used to insert something behind the platen to trigger the paper sensor so that it would let my Apple go.

But my printer also supports an "ignore paper error" command. The only trick was getting AppleWorks to send the code. I decided to put the code in an unused characters-per-inch setting. My idea was to switch to 24 characters-per-inch at the beginning of a letter, send a blank line, then switch to the character-per-inch setting I really wanted to use. But AppleWorks wouldn't send the code for 24 characters-per-inch (really the ignore-paper-error code, of course) unless I put something on the blank line besides spaces or a carriage return. Since anything else would get printed on my letter, this was a significant problem.

I finally discovered that a "sticky-space" (created with open-apple-spacebar) was the solution I needed. A sticky-space won't print anything, yet it forces AppleWorks to send the character-per-inch code. Other people trying to use a custom printer's character-per-inch codes for unsupported printer features may find the sticky-space helpful.

Paul Lucas
Levittown, N.Y.

Keyboard problems explained

David Grigg's problem with spurious characters in AppleWorks (October 1986, page 2.69) is a hardware problem all right—and it's not limited to AppleWorks, or even to his IIc. Dave Cortesi discovered a similar problem a few years ago on a different computer, and reported on the problem in his ***Dr. Dobb's Journal*** column.

The spurious characters appear because of the way that the Apple keyboard's switches are networked. It's like a grid of horizontal and vertical wires, which each key at an intersection. The grid looks something like this:

```
e 1 2 3 4 5 6 7 8 9     e is escape
t Q W E R T Y U I O     t is tab
A D S H F J G K ; L     c is return
Z X C V B M N , . /     x is delete
          \   = 0 -     s is spacebar
          '   P [ ]     u is up arrow
          c   u s '     d is down arrow
          x   d l r     r is right arrow
                        l is left arrow
```

The system works fine for detecting one key at a time and it successfully filters out most cases where more than one key is pressed at a time. But when three keys pressed form three corners of a rectangle on the grid, the key at the fourth corner of the rectangle will also appear to have been pressed. For example, hold down the H, E, and R keys on a IIe or IIc keyboard and you'll get a spurious F on the screen. If you have a Laser 128 or a IIgs you'll have the same problem, but a different grid layout.

So it's a hardware problem—and the only answer is to make the human at the keyboard adapt. Lift your fingers when you type.

Frank Hayes
II Computing
San Francisco, Calif.

Softdisk lives

If *Softalk* "died" then does that mean *Softdisk* has "died" with it?

R. Roehm
Los Alamitos, Calif.

***Softdisk**, a monthly Apple II magazine-on-disk, is alive and well. You can contact the company at PO Box 30008, Shreveport, LA 77130. Prices are $49.95 for a 6-month subscription, or $89.95 for a 12-month subscription. Each monthly issue includes two disks.*

*A similar publication is **UpTime** (174 Belleview Avenue, Newport, R.I. 02840). The cost for **UpTime** is $66 for 12 months (1 disk/month).*

Readers beware

Both of the methods for embedding machine language code in Applesoft programs mentioned in the September 1986 issue of Open-Apple ("Another place for code" page 2.63) are in danger from certain utilities. Obviously, a REM stripper will wreck havoc with machine language code inserted in REM statements. And most RENUMBER utilities will reset the end-of-program pointer back to where it "belongs," thus deleting any code appended to the end of an Applesoft program.

Regarding "How many drives have we here? (November 1986, page 2.80): You *can* tell the difference between no drive and no disk with the UniDisk 3.5 (and Apple 3.5 on a IIGS). The 3.5 inch drives return the offline error ($2F) with no disk inserted. Even though this is not a defined error under ProDOS, the Kernal passes it on to Basic.System. Unfortunately, it is not pigeon-holed in a global page location and it is converted by Basic.system into the umbrella I/O ERROR. If you're willing to search for it, the ProDOS error number can be found in a scratchpad location —in the IIc at $4F8; in the IIe at $4F8 + the number of the slot the interface card is in; and in the IIgs at $E00FB1.

In your answer to that same letter, you state that third and fourth drives hooked to one controller are "supposed" to appear to be in slot 1 or 2, but Apple has never suggested that higher slot numbers can't be used. However, the Smartport interface will automatically access the third and fourth devices in one of its daisy-chains only if they have been assigned a phantom slot lower than four.

Speaking, finally, of four drives, you can easily patch FID, as came up in October (page 2.71), to handle up to nine DOS 3.3 drives. Here's how you do it:

```
DRIVES = 4
BLOAD FID
POKE 2419,176 + DRIVES + 1
POKE 2495,176 + DRIVES + 1
IF DRIVES > 7 THEN POKE 2431,15:POKE 2507,15
BSAVE FID,A$2051,L$4687
```

Dec. No's

Tom Vier
Reston, Va.

Another restart trick

Here's another trick like the one for restarting AppleWorks that you published in June (page 2.33). If you crash into the Monitor from a ProDOS system file, there is usually no way to restart except by rebooting. Basic.system supports the old 3D0G warmstart vector, but few other system programs do.

However, if you reboot ProDOS, it will erase any files you have stored on the /RAM volume. Instead of rebooting, enter the following. It will execute a ProDOS quit call. If you have a program selector such as ProSel, it will run automatically. Otherwise, you will at least get the ProDOS quit prompt.

```
800:20 00 BF 65 07 08 00 04 00 00 00 00 00 00 N 800G
```

The source code looks like this:

```
20 00 BF   JSR  $BF00      ; call MLI
65         DB   $65        ; MLI QUIT call
07 08      DW   *+3        ; pointer to parm table
00         BRK             ; BRK if ProDOS won't quit
04         DB   $04        ; parm table - 4 parms
00         DB   $00        ; all 4 parameters are $0's
00 00      DW   $0000
00         DB   $00
00 00      DW   $00
```

I have found this trick very helpful while programming in ***Kyan Pascal***.

Jim Luther
Kansas City, Mo.

Users battle tech support

If you charged for your newsletter according to what the information was worth, I'd have to cancel all of my other Apple magazine subscriptions, pawn my wife and kids, and maybe even put off buying a new IIgs. Once more, your November issue proved that **Open-Apple** is an absolutely indispensable source of solid advice.

To wit, I had been experiencing track 0 disk crashes when using ProDOS on a floppy drive for nearly six months. The problem began when we purchased a new IIe system at work, complete with an Applied Engineering TransWarp card. After purchasing two surge protectors, replacing the power supply with a heavy-duty unit, swapping out all peripheral cards, exchanging disk drives, and purchasing an expensive disk drive analysis program, I felt I had traced the problem to the TransWarp card. The disk erasures only occurred when using a ProDOS-based program on one of the floppy drives and only when the TransWarp card was active. The problem never occurred under DOS 3.3, never involved the UniDisk 3.5 drive, never affected the IIc system plugged into the same power strip, and never happened when the TransWarp card was either removed from the computer or deactivated.

With the TransWarp activated, the problem occurred approximately every third disk access. Furthermore, the problem occurred with varying frequency in any of three IIe systems that the TransWarp card was plugged into. Pretty convincing evidence implicating the TransWarp card, wouldn't you agree?

Applied Engineering didn't agree. I exchanged the card twice with them, wrote detailed accounts of my bug-hunting treks, and paid for three lengthy phone calls to their technical assistance line in Texas. On each occasion, I was told that the problem could not be replicated and that the problem was unequivocally a "user error." When I mentioned that your newsletter had been carrying on-going descriptions of the problem since January, their terse response was that Applied Engineering receives hundreds of calls each day so if there was any problem they would know about it.

In mid-November, I called them once more to read select portions of your article "ProDOS bug found in Australia" (November 1986, page 2.73). The technician's reply was that the article was simply wrong and that the problem could only be user-related. I thanked him for his input and requested the return of my TransWarp card. After reinstalling the TransWarp a few days later, I verified that the track 0 problem was still occurring and then made the ProDOS patches outlined in your newsletter. Voila! We have not lost a single data disk since.

What I have learned from this experience is that some companies work under the assumption that their products are bug-free but that their customers are idiots. Perhaps Applied Engineering should stop acting like it's God's gift to the Apple user and spend a little time debugging their "public interface."

Thanks for your fabulous publication. And, most especially, thanks for communicating with your readers from a position of respect and appreciation. I don't understand some of your articles the first time I read them, but I have never felt intimidated by your descriptions nor belittled by your language. Maybe that's why I genuinely enjoy re-reading the articles time and again.

Pat Marnett
Tempe, Ariz.

***Open-Apple** is always happy to play the role of the hero, but I think we should point out that the credit here goes to other people. Subscriber J. Ernest Cooper was the one who convinced me the problem was more than "user error" and who got me to report the problem in **Open-Apple**. Stephen Thomas was the Australian programmer who listened carefully to his customers' feedback and who consequently found the ProDOS bug. Subscriber David Grigg made the connection between what Cooper had reported and what Thomas had written for an Australian user group newsletter, and who **wrote us** about it. This kind of world-wide information exchange is the core around which **Open-Apple** is growing.*

Having been on both ends of technical support lines, I think you're being too harsh with Applied Engineering. As it turned out, after all, fixing ProDOS, not the TransWarp, is what solved your problem. On the other hand, any company that gets hundreds of support calls a day should have gotten enough reports of track 0 crashes by now to suspect something was amiss. Technical support people have got to be more cautious with exotic problems. It's easy to blame bug reports that you have no solution for on "user errors" and forget them.

For example, my own brother-in-law called me this week to ask why so many of his AppleWorks data disks were being destroyed. When I told him I had a fix for him, he said, "That's funny, last year you told me it was my fault." I don't even remember talking to him about crashed data disks before.

How come, by the way, we're not laying any blame on Apple's shoulders? It's Apple's own operating system that has been crashing all these data disks after all. However, Apple's technical support group has a world-wide network of dealers that ensure it will never hear about, much less find, any bug that has even the slightest chance of being blamed on "user error."

*For more perspective on the battle between troubled users and defensive technical support see the note from **Call -A.P.P.L.E.'s** Charlie Stillman on page 55 of the November 1986 issue of that magazine. "Pinpoint was driven to the edge by Dennis LaMonica's report of a serious bug in Pinpoint's **Graphmerge** accessory in the July 'Write A.P.P.L.E.' Outrage, hurt, and denial eventually gave way to a patch to fix the problem....The Pinpoint technical staff is courteous, responsive, and effective....But Pinpoint almost wrote off Dennis LaMonica as a trouble maker. Publishers need to recognize the LaMonica's of the world as their best allies. For each LaMonica who takes the trouble to report a bug, there are ten other people who quietly file the software in a drawer and spread the word that it's unreliable. Keep up the good work Pinpoint and LaMonica! You are on the same team!"*

Open-Apple

is written, edited, published, and

© Copyright 1986 by Tom Weishaar

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

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

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

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

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

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

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

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

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

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

Source Mail: TCF238 **CompuServe: 70120,202**