August, 1990
Volume 1, No. 6

8/16

The Journal of Apple II Programming $3.50

Toto, I don't think we're in Kansas anymore.

In this issue:

Ariel Publishing
P.O.Box 398
Pateros, WA 98846
(509) 923-2249

Slide-On kits are $14.95 ea.
$12 ea. in quantities of 10+.

New kit restores your Apple IIGS

If you purchased an Apple IIGS computer before August 1989 (512K model), a Lithium battery was soldered onto the computer board at the factory and the internal clock started ticking. It is just a matter of time until the battery runs out of juice and your computer forgets what day it is and any special settings you have selected in the Control Panel.

If the software you are running uses the date and time to keep track of records you could be in for real trouble when the clock runs out. The IIGS is also known to lose disk drives along with numerous other side effects caused by a dead battery.

Before the introduction of Nite Owl's Slide-On battery, the normal method for replacing the IIGS battery was to pack your computer up and take it to your local Apple dealer. That was very inconvenient, time consuming, and expensive for the typical computer owner.

Slide-On battery replacement is not much more difficult than changing a light bulb. Using wire cutters, scissors, or nail clippers, the old battery is removed leaving the original wires still soldered to the mother board. The new Slide-On battery has special terminals which have been designed to fit onto the old battery wires. It usually takes only a couple of minutes. Complete, easy-to-follow instructions are included with every kit.

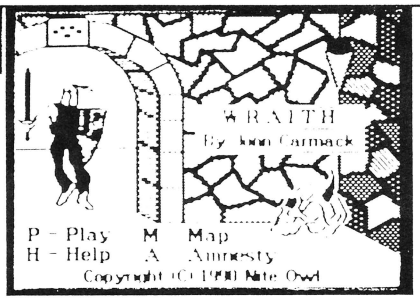
Typically, our customers have reported that the original equipment batteries have an average life expectancy of 2 to 3 years. This is about half as long as they were supposed to last. Slide-On replacement kits include Heavy Duty batteries which should provide for a longer battery service life.

We highly recommend that every IIGS owner keep a spare battery on hand, ready for when the inevitable battery failure occurs. These Lithium batteries have a shelf life of over 10 years, and come with a full 90 day satisfaction guarantee.

New

WRAITH
Adventure Game

Special Introductory Price $9.95*



This graphic adventure game comes complete on a single 3.5 inch disk with on-screen instructions, a map, demo play option, and dungeons which were too vast and expansive to fit on 5.25" disks.

The object is to search out and destroy the evil WRAITH to save the mythical island of Arathia. To succeed at this quest the adventurer must fend off many monsters, learn magic spells, and buy weapons and armor to defeat the evil WRAITH.

Works on ANY Apple][with a 3.5" drive. It will have a retail price of $14.95. One of the best software values ever! * Offer expires 12/31/90

Font Collection *- The A2-Central staff has spent years searching out and compiling hundreds of IIGS fonts. These fonts are packed onto eight 3.5 inch disks. They work with IIGS paint, draw, and word processing programs. Includes a program to unpack them and an Appleworks data file.* $39

School Purchase Orders are Welcome.

Ship to:

Telephone #:

• Bill To •

Cash, Check, Money Order

VISA

Master Card

Purchase Order

Credit Card or PO#

Expiration Date

Quantity	Description	Price	Amount
	Slide-On Battery Kits	$ 14.95	
	WRAITH Adventure	$ 9.95	
	Font Collection	$ 39.00	
Signature for Credit Card Orders		Kansas Sales Tax	
Please include $2 shipping and handling / $5 for overseas orders. Kansas residents add 6% sales tax.		Shipping & Handling	
		TOTAL	

Please give us a call today at: **(913) 362-9898**

Photo-Copyable FAX: (913) 362-5798

Nite Owl Productions
5734 Lamar Avenue A
Mission, KS 66202
USA

(Cut & Paste Address Label)

Prices may Change without notice.

8/16

Publisher & Editor-in-Chief	Ross W. Lambert
Classic Apple Editor	Jerry Kindall
Apple IIgs Editor	Eric Mueller
Contributing Editors	Walter Torres-Hurt
	Mike Westerfield
	Steve Stephenson
	Jay Jennings
Subscription Services	Tamara Lambert
	Becky Milton

Introductory subscription prices in US dollars:

• *magazine*
1 year $29.95 2 years $56

• *disk*
1 year $69.95 6 mo $39.95 3 mo. $21

Canada and Mexico add $5 per year per product ordered. Non-North American orders add $15 per year per product ordered.

Direct all correspondence to: Ariel Publishing, Inc., P.O. Box 398, Pateros, WA 98846 (509) 923-2249 (voice), (509) 689-3136 (fax)

We here at Ariel Publishing freely admit our shortcomings, but nevertheless strive to bring glory to the Lord Jesus Christ.

The Publisher's Pen

by Ross W. Lambert

I have finally entered the publisher's time warp. I am writing for a post- A2 Central Summer Conference audience even though the event has not happened yet. For that reason we'll be covering the happenings in more detail next month. About all I can tell you now is that by the time you read this Apple will have unveiled (but not officially released) some remarkable achievements in Apple II software engineering. Probably the most remarkable is HyperCard GS. If the rumors are correct, this critter can run stacks written on the *Macintosh.* I bet somebody suffered to make that happen.

According to the conference schedule, we'll also get a look at an animation toolkit for the GS and a few other goodies. I may not be able to discuss *everything* in great detail because many of the Apple, Inc. seminars require non-disclosure agreements. Still, we'll bring you all the news that fits in print, or whatever.

So hang on until next month. I bet we scoop *inCider/A+.*

Half a birthday to you, half a birthday to you...

8/16 is one half of one year old with this issue. I'd like to use this occassion to humbly request your analysis of our publication. We have included a subscriber survey form toward the back of this issue which is designed to be cut out. There is no "content" on the back side of it or anything.

There are lots of questions on the form, some of which are about you. Our intent with this survey is not to be nosey, but rather to A) massage our content to meet your desires, and B) get a demographic profile of our subscribers. Let's be frank: advertisers like that sort of thing. Even so, feel free to skip any part of the form that you don't want to fill out.

This magazine is not nearly as dependent on advertising as most publications, but we'd be fools not to encourage folks to travel that avenue.

WARNING: Let me *strongly* encourage you to fill out the survey. These types of things tend to attract a disproportionate number of the "least satisfied". If you are pretty much happy with the magazine it is important that you say so in order to help prevent us from having a skewed view of the results. Ariel Publishing, Inc. is most definitely a customer-driven sort of operation, so if folks seem to be saying "Jump!", our response will be "How high?"

The View From Pateros

At the risk of sounding arrogant, nearly 100% of the mail we've gotten has suggested that *8/16* is the best thing since frozen yogurt. I am proud of our work to date, yes, but I know it is far from perfect. And a few folks have shared their concerns with me already. Here's where ***I*** plan to put our emphasis in the future:

❑ More graphical presentation of concepts (a picture is worth a thousand words, so to speak). This is harder than it sounds.

❑ More and lengthier explanations. Some of our authors (including me) have made some pretty hefty conceptual leaps at times.

❑ More code in higher level languages (C, Pascal, Micol Advanced Basic).

❑ Earlier distribution. We're moving back production for the October issue by one full week. This should get it to most of you by the first day or two of the month.

❑ Faster, more consistent service. We've hired two part time people to help fill your orders and manage our data entry. Such expansion is a fairly bold move in the face of the current Apple II market. We are *not* a mail-order software house, however. I do *not* plan to aim for the 24-48 hour turnaround of the mail houses. I *do* plan to get orders filled in approximately five working days or better.

Those are my priorities for our second six months of life. Now it is your turn to tell me yours.

== Ross ==

IIgs Programming

Multi-Bank: A IIgs Monitor Utility

by James Hodge

Editor: We here at 8/16 are quite pleased to have James write for us. As a long time CALL A.P.P.L.E. author he has already wowed us with neat stuff on many occassions. Welcome!

Modern computers are really something! Long gone are the days when a 48K Apple was considered a big machine. Now you need 768K to 1280K to run many common games and applications. To be fair, a lot of the software available today is pretty impressive, despite the hardware requirements.

Monster machines place an added burden on programmers. It would have been nice if Apple had endowed some of the IIgs monitor commands with the ability to work across bank boundaries, but they didn't. As a result, trying to clear out 10 or 12 banks of memory, or find a byte pattern "needle" in the memory "haystack" can be a tedious task. The "Multi-Bank" monitor utility presented here can ease some of the tedium by forcing the Apple to do more of the work.

The Multi-Bank routine takes advantage of what is probably the least used feature of the Apple monitor, the ability to repeat some, or all, of the command line. This is documented in the old _Apple][Reference Manual_ on page 56 and in the new _Apple IIGS Firmware Reference_ on page 46.

As explained in the Firmware Reference, the repeating command trick alters the index that the monitor uses as it scans the keyboard input buffer at $00/0200. When the monitor locates and executes a command it saves the value of the index at $00/0034 (also called YSAV) and then reloads the index register and resumes scanning the command line when it has finished. By changing the value at $34, you tell the monitor to continue parsing the input at some other point in the keyboard buffer.

To see what happens to location $34 when the monitor parses the command line, try the following example (tilde characters (~) are used to indicate spaces):

```
*n~~34~~34~~34~~34:0~<ret>
```

The result will look like this:

```
00/0034:06-.
00/0034:0A-.
00/0034:0E-.
```

This will repeat until you press Control-Reset or shut the machine off.

To make use of this trick there are a couple of rules that must be observed. You need to start the repeating portion of the command line with a letter command (N, the monitor "normal video" command, is suggested) and end the command line with "34:x ", where x is a hex value specifying the position of the start of the loop. To start the loop at the beginning of the command line, x would equal 0. The line MUST end with a space. The manuals state that the only way to end the loop is by pressing Control-Reset.

"The Multi-Bank routine takes advantage of what is probably the least used feature of the Apple monitor, the ability to repeat some, or all, of the command line."

With a little extra machine language, the "repeat" command becomes quite a time saver. Multi-Bank,

a routine that runs at $300, allows monitor commands to work throughout the memory of a IIgs, rather than being limited to a single bank at a time. The upper limit of memory you wish to operate on is hard-coded in the program, and when that limit is reached Multi-Bank will terminate cleanly. Multi-Bank was designed to work in conjunction with the pattern search (P) and fill memory (Z) monitor commands, but it will also work with the move (M) and verify (V) commands. It's also possible to arrange a command line to make Multi-Bank work with the memory change and examine commands.

About Multi-Bank

On the first pass through Multi-Bank, an indirect address pointer to the input buffer is set up at locations $8 and $9 and a flag is set to indicate that initialization has been performed. Multi-Bank then searches for, and expects to find, the "<" character within the first $40 bytes of the input buffer. If the "<" symbol isn't found in the first $40 bytes of the input buffer, the command sequence will terminate. Assuming that it is found, Multi-Bank leaves the pointer aimed at the two digit bank specification that follows it. The initialization step will not be used again. At this point Multi-Bank increments the bank number and checks to see that it is within the limit set by "max_bank". If it is, control returns to the monitor and the rest of the command line will be scanned and executed. If the bank number equals max_bank, Multi-Bank will re-zero the "frstpass" flag and place a Return character in the command line to end the loop.

Multi-Bank expects to be called with the registers set to their 8 bit width. Adding the command "SEP $30" at the beginning of the routine would ensure the condition was met, but right now it is the user's responsibility. Multi-Bank will only work on the bank number following the first "<" symbol in the command line. The routine is machine-specific since the value for max_bank is hard-coded. The value for max_bank should equal the highest bank number plus 1. Different users will have to determine the upper limit that suits their needs. Generally, you won't want search (P) or zap (Z) commands operating on the so called ROM disks provided by battery backed up memory (e.g., the Applied Engineering Ramkeeper) or in memory allocated to RAM disks. As with any monitor command that specifies a range of memory to work with, you MUST be careful not to reference the areas of memory containing the softswitches (unless you like to turn on disk drives and change the screen mode from text to trash).

Using Multi-Bank

Installing Multi-Bank might take a little forethought. If you are running ProDOS 8 or DOS 3.3 then you can BLOAD MULTI.BANK. If, while in a GS/OS application, you expect to get into the "Visit Monitor" desk accessory and use Multi-Bank you'll want to put the program on a RAM or ROM disk, or store it in a (hopefully) unused area in memory, and then use the memory move (M) command.

There are times during program development that I need to know what areas of memory are being used. Since I have an AE RamKeeper installed in my system, it causes "clutter" to accumulate (old, junk values in memory) that won't go away by simply switching off the machine. A command to set most of the memory in a IIgs to zeroes would look like this (again, tilde characters (~) are used to indicate spaces):

```
*n~~0<02/0.ffffz~~n~~0/300g~~n~~34:0~<ret>
```

This example would first set bank 2 memory to zeroes and then it would call the Multi-Bank routine with the command "0/300g". Multi-Bank would do its work and return control to the monitor. The rest of the line would be parsed and executed and, since $34 had been set to 0, the command line would execute again, but with a bank value of 03. This process would repeat until Multi-Bank reaches its limit, at which point it would replace the next command in the line with a Return character to force the loop to terminate. The example starts at bank 2 so it will avoid the $C000.C0FF softswitch areas in banks 0 and 1. (Warning: This command string can destroy the operating system in RAM memory. GS/OS system 5.0 loads into RAM so that switching between Applesoft and P8 applications and GS/OS applications goes quickly. If you launched BASIC under GS/OS, you may need to switch your GS off and on again to reboot your system.)

Using Multi-Bank with a pattern search (P) command requires extra care in the way the command line is formed. When the monitor processes a search request it places a length byte at $200 followed by the pattern to search for. That destroys the first part of the command in the input buffer. The workaround for this problem is to start the command line with blanks or "N"s (Normal video commands). There should be enough of a buffer to allow for the search pattern and its length byte. The value to store at $34 should point to a letter command at the end of the pattern buffer area. I usually leave an area slightly larger than I need, rather than trying to figure out the smallest space needed. Multi-Bank pattern searches look like this:

```
*~~~~~~~~~~~n~~\6b 08 e2\<02/
0.ffffp~~n~~0/300g~~n~~34:b~<ret>

*nnnnnnnnnnnn~~\"hi world"\<00/
0.bfffp~~n~~0/300g~~n~~34:b~<ret>

*n~~~~~n~~\"Apple"\<02/0.ffffp~~n~~0/
300g~~n~212.215~~n~~34:6~<ret>
```

The first two examples show command lines with buffer spaces larger than needed. The first searches for a three byte pattern, and the second example searches for the text "hi world". (Note: The high bit in the bytes of the text pattern will depend on the system mask (F). Using the monitor command "FF=F" before entering text into the monitor will set the high bit, while the command "7F=F" will clear it.)

The third example does a pattern search and calls Multi-Bank, but before it repeats it dumps memory in the range 0/212.215. This allows you to see the bank values change, but it also clutters up the display and makes it harder to see the output from the search command.

It's possible to enhance the memory change command to work across bank boundaries using Multi-Bank with a creative command line. The following example shows how to place a particular byte sequence into multiple memory banks:

```
*34:1a~~n~~<01/1000:1~2~3~~n~~0/
300g~~n~~0/20a:a0~~n~~34:7~<ret>
```

The command "34:1a" causes the monitor to skip the "n <01/1000:1 2 3" and go to the second "n" in the command line (at $00/021a). In this case the index at $34 is adjusted to point farther into the input buffer, rather than towards the buffer start. When Multi-Bank is called it sets up a pointer to the bank specifier following the "<" character and then increments the bank number. The next significant command, "0/20a:a0", changes the "<" character to a blank. The "<" character is needed only so Multi-Bank can find the bank number. Control then loops back to repeat the command line, starting at the normal video command "n" at $00/0207. When the monitor executes this line the second time the command "<01/1000:1 2 3" has been changed to " 02/1000:1 2 3", so the values 1, 2, and 3 are placed into memory starting at $02/1000. Multi-Bank increments the bank number again and the process repeats until the max_bank value is reached. Because this sequence of commands increments the bank number first, you need to start with a bank value of one less than the first bank you want to affect.

To summarize the rules for using Multi-Bank:

✔ 1. It must be located at $0/300, unless it is "ORGed" and assembled to execute at a different address.

✔ 2. The value of max_bank should be set to either the highest bank number + 1 of the machine, or to the value of the highest bank + 1 that you wish to operate on, as appropriate.

✔ 3. Max_bank is encoded as a pair of ASCII digits with their high bits set. Valid values range from "0" through "9" ($B0 to $B9) and "A" through "F" ($C1 to $C6).

✔ 4. There must be at least one blank following the command to set $00/0034.

✔ 5. The firstpass flag should be zero. If a previous command terminated prematurely, either by hitting reset or by failing to include a space after the command "34:x", you will have to zero the flag or reload a fresh copy of Multi-Bank.

✔ 6. The registers should be set to 8 bit width. The value of the m and x flags should be 1.

✔ 7. Multi-Bank expects to find the "<" character in the first $40 bytes of the input buffer, followed by a 2 byte

bank specification.

While Multi-Bank may not revolutionize your development work, it should prove itself useful if you need its ability.

Program Listing:

```
*   Monitor Command Multi-Bank Looper
*      by James A. Hodge - 9/19/89
*        Assembler: Merlin 8/16
* Copyright (c) 1990 Ariel Publishing
* and James Hodge - some rights reserved

         tr    adr
         org   $300

         lda   frstpass ;first pass flag
         bne   do_it    ;do fndbnk 1st
                        ;pass only
         inc   frstpass ;set flag
         lda   #2
         sta   $9
         stz   $8  ;point 8.9 at kbd buff

findbank inc   $8  ;increment pointer
         lda   $8  ;check it
         cmp   #$40
         beq   so_long;quit after X bytes
         lda   ($8)     ;locate 2 byt
                        ;bank# that
         cmp   #"<"     ;after "<" symbol
         bne   findbank ;if not "<"
                        ;continue
         inc   $8 ;leave this -> to bank#

do_it    ldy   #1
         lda   ($8),y
         inc      ;inc low byte of bank#
         cmp   #"9"+1 ;valid values =
                      ;0-9, A-F
         bcc   o_k    ;if val = 0 to 9
         bne   nextchar
         lda   #"A"    ;"A" = "9" + 1
         bra   o_k
nextchar cmp   #"F"+1
         bcc   o_k     ;if val = A to F
         lda   ($8)    ;else
         inc           ; inc high byte
         sta   ($8)
         lda   #"0"    set lo byte to "0"

o_k      sta   ($8),y ;now bank#'s inc'd
         cmp   max_bank+1;chk lo byt
                         ;of max_bank
         bcc   go_on
         lda   ($8)    ;check hi byte
         cmp   max_bank ;if bank# >=
then quit
         bcs   so_long
go_on    rts

so_long  stz   frstpass ;0 flag for reuse
         ldy   $34   ;get cmnd line index
         lda   #$8D ;carriage return char
         sta   $201,y ;terminate cmd line
         rts

max_bank asc   "18"
frstpass hex   00
```

The ZBasic Zealot

Checking Out the Locals

by Ross W. Lambert

You Z-fans out there got a little bit short-changed last month; I had to cut an article due to lack of space. I ended up axing my own contribution. I thought I'd counter that this month by providing you with an extra dose of Z-Power. We're going to cover two separate topics; hang on to your hats. Those of you who are not really into ZBasic may want to read along anyway because I dip into compiler construction just a tad. You might (shudder) learn something.

Part I : FN Local and FN Global

Those of you who read these articles closely probably noticed that I promised local variables for ZBasic in a "teaser" at the end of my June column. From the sounds of things, nobody really believed me.

O ye of little faith.

No, I didn't rewrite the language or even patch it. Rather, I created a little function that takes advantage of the way ZBasic organizes data in memory. In a nutshell, I stash the variables you want "localized" into a buffer. When you want their values restored, I move them from the buffer back into the correct spots in memory.

"You can access the Apple monitor from within a running ZBasic program."

Easy stuff. And it was made even easier by a fact that may surprise a few of you; you can access the Apple II monitor from a running ZBasic program. Applesoft is long gone, of course (i.e. switched out), but you can still get at the system monitor in the highest reaches of ROM.

Inside the monitor ROMS there lives a little memory move routine. It hides discreetly at location $FE2C. The only caveats for using this bugger are that it will not copy data between banks (main and aux) and your source block cannot overlap to the right (i.e. to the high memory side) of the destination block (see Figure 2). Neither caveat is an issue for our current purposes. We only intend to move memory in main memory, and our source variable data and the data buffer are in separate, discrete blocks of memory.

Speaking of blocks: that brings us to the costs of FN Local.

There ain't no free lunch

Here's a mini-lesson on compilers... in other times and places (and CPUs) compilers create what is called a "stack frame" whenever they encounter a function, subprogram, etc. A stack frame is simply a little section of memory on the stack (c.f. my June

'90 column) that belongs to the function being called. The first thing a compiler usually does when creating function code is to save the current address of the stack pointer. The function is then allowed to keep its variables on the stack, and then when the function is all done it just resets the stack pointer to its old value. The function's old local variables just float away.

When I first attacked the question of local variables, I'm sure that I had to grapple with the same issues the ZBasic authors did. On most microcomputers, and especially eight bit Apple IIs, stack space is at a premium. We've only got 256 bytes, and most of that will already be spoken for by the time one of our functions steps up to bat.

Some compilers have opted to create what is often called a "pseudo-stack". This section of memory behaves like the normal system stack, but the CPU itself does not manage it or have instructions to access it directly. A pseudo-stack is merely a Last In First Out data structure managed by the language, not the CPU. They are just like the LIFO structures I mentioned in my June '90 column.

One problem with pseudo-stacks is that they tend to be slow. Another problem is that they take up memory, and usually a lot of it. ZBasic does indeed have a pseudo-stack for its various internal operations, but it is relatively small because it doesn't mess with local variables.

This is all a long winded excuse for not creating a FN Local that pushes your local variables on the stack. Chances are you'd get a ?STACK OVERFLOW error or trash the system so often it'd be more trouble than it is worth.

The simplest way to provide some of the functionality of local variables without hardly any hassles at all is to ask you to plan ahead. Thus FN Local comes at two main costs, organization and memory.

The organization cost involved stems from the fact that the range of variables you'd like localized *must be contiguous in memory.* In order to *make* them contiguous, you have to dimension them in order. For example,

```
DIM IntOne, IntTwo, ArrayOne (999), IntThree
DIM 100 MyString$, IntFour
```

...creates a block of memory 2108 bytes long. Each integer variable is two bytes long (for a total of eight bytes), the string is defined to be 100 bytes, and a 1000 element integer array at 2 bytes per element is 2000 more bytes. If I added correctly, we get 2108.

You could, using FN Local, save this entire block of memory, pollute each and every one of the variables, and then restore them all in one fell swoop with FN Global.

Neat, huh?

Well, yes, except that there is one more cost: RAM. Since I have elected *not* to push the variables to be localized onto the stack, you must provide some RAM for temporary storage. Since every program (and indeed, every programmer) is different, I decided not to mandate a buffer size or location. That is up to you. Fortunately, there are several potential locations that may be useful depending on the type of program you are writing:

❑ One obvious option is to create your buffer in the standard ZBasic data space (in main mem). This is what I chose to do in the demo code, DIMensioning a LocalBuffer(10) array. This storage option works best when you don't have tons of variables to localize or you have lots of data storage space available.

❑ Another option is to use the high resolution graphics page. This whopping 8K block ($2000-$3999) is unused by text based ZBasic programs.

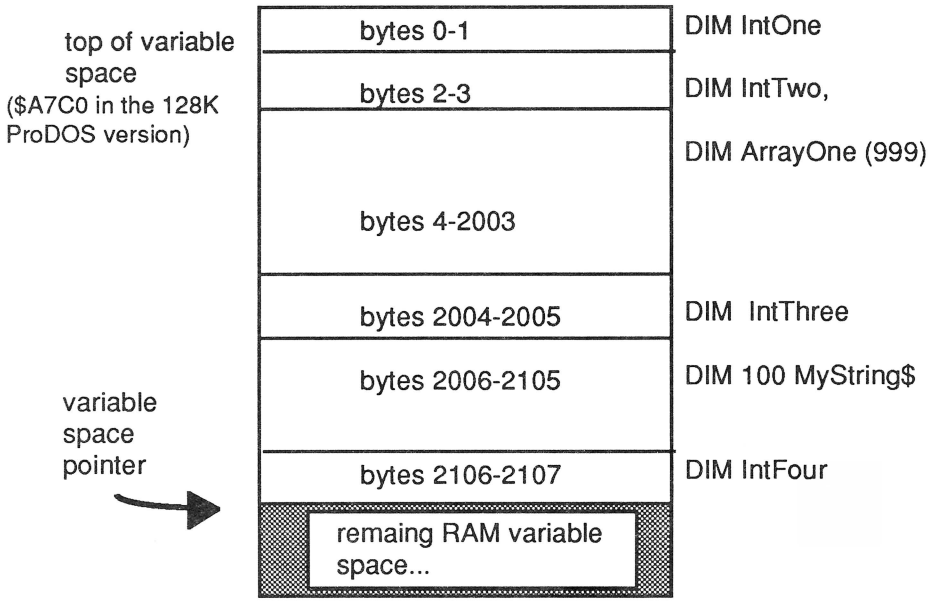
❑ There is also the page three free space (200 bytes starting at 768).

❑ Finally, you could find the start of your block with VARPTR, save it to a temporary file on a disk or RAMdisk with the BSAVE function (on the ZBasic distribution disk), and then restore the entire thing with FN BLOAD. This would, in fact, create something a little bit akin to virtual memory. It's really a neat trick for super memory intensive applications.

Back to the BASICs

When a new variable is DIMensioned or defined in ZBasic, the variable storage pointer is moved *lower* in memory by the appropriate amount (c.f. Figure 1).

Figure 1: ZBasic Variable Memory Use



This means that if the first four variables you define are integers, you can find the beginning of the 8 byte block they occupy by getting the VARPTR of the last variable defined. Like so:

```
DIM One, Two, Three, Four
BlockStart = VARPTR(Four)
```

The same principle applies to all data types; integers are just easier for demonstration purposes. Consult your ZBasic manual for the lengths of other data types (and don't forget to consult your configuration screen for how many digits of precision you've set for floating point vars).

Don't let arrays throw you off either. Although they store their data *internally* lowest to highest in memory (i.e. element one starts *lower* in memory than element two), the start of the entire array is *lower* than the variable declared or DIMmed immediately before it.

How to get it to do what it does

As I mentioned earlier, you two main concerns when localizing variables are "what" (what variable block to localize) and "where" (where to put it). Assuming you can answer both of those questions, the syntax to the function looks like this:

```
REM saves bytes starting at Source
FN Local (Source,Bytes)

REM restore bytes that live at Source+
FN Global (Source,Bytes)
```

As you might guess, one important use of these functions is inside other functions. For example,

```
LONG FN MyFN
   FN Local(Source,Bytes)
 ... pollute all the variables living at
Source to Source+Bytes
   FN Global (Source,Bytes)
END FN
```

This means that you can really make your functions independent of one another and increase the liklihood that you can use them in another program without variable trashing.

Another approach, and the one I used in the demo, is to call FN Local *before* calling a function and FN Global *after* calling a function, thereby restoring any variables that may have been trashed during the passing of parameters. Like so:

```
FN Local(Source,Bytes)
X = MyFN (Whatever)
FN Global (Source,Bytes)
```

Digging into the Demo

You'll notice that the functions assume that the start of the variable buffer can be found at LocalBuffer.

You assembly hacks will also note that I set the Y register of the CPU to zero before the jump to $FE2C. This is necessary because Y is used as an offset for the indexed indirect addressing loop that lives in the monitor memory mover. And I "bounced off" the RTS in ROM, saving a byte (I'm cheap), hence the JMP instruction in the MACHLG statements and not a JSR.

The LONG FN MoveMem does most of the work; FN Local and FN Global switch the parameters around for you so that it is all a little less taxing on the ol' noggin.

Improvements

Now that I've pretty much finished the code and run up against our publication deadline (earlier this month than normal due to KCFest), I've thought of some really wonderful additions...

As the functions stand right now, they don't manage the buffer memory for you at all. They always stash the data to be saved at the begnning of the buffer, memory location LocalBuffer. A better solution would allow you to stash as many sets of data as would fit into the buffer. This would involve keeping track of what was where, but that could be easily handled with a BuffPtr(x) array. For the block of data you number as X, look up its position in the buffer by reading BuffPtr(X).

Another approach would be to treat the data buffer as a pseudo stack. Each time you "push" a group of values onto the stack, you increment the "stack pointer" by the size of the block. To get the values back, just "pull" the values off the stack and down into the right location in memory.

Hopefully this code will inspire you to do something truly outstanding, or at least make your life a little easier.

Listing 1: FN Local and FN Global

```
REM ================================
REM FN Local and FN Global
REM
REM by Ross W. Lambert
REM Copyright (C) 1990
REM Ariel Publishing, Inc.
REM Most Rights Reserved
REM
REM I assume that the default
REM var type is integer
REM ================================
 :
 DIM Xtop,Ytop,Xbot,Ybot : REM demo data
 :
 DIM LocalBuffer(10): REM temp buffer
 :
 ML = 768
 LocalBuffer = VARPTR(LocalBuffer(0))
 :
 LONG FN MoveMem (Source,Dest,Amount)
   Z1 = PEEK WORD (60) : REM save zpg
   Z2 = PEEK WORD (62)
   Z3 = PEEK WORD (66)
   :
   POKE WORD 60,Source : REM  poke addrs
   POKE WORD 62,Source+Amount
   POKE WORD 66,Dest
   :
   REM This ML routine could be
   REM deposited at startup and
   REM then just CALLed as long as
   REM it was not overwritten by
   REM anything else.
   :
   REM ldy #0
   POKE ML,160:POKE ML+1,0
   REM jmp $FE2C
   POKE ML+2,76:POKE ML+3,44
   POKE ML+4,254
```

```
  CALL ML
  :
  POKE WORD 60,Z1 : REM restore zero pg
  POKE WORD 62,Z2
  POKE WORD 66,Z3
END FN
:
:
DEF FN Local (Source,Bytes) = FN MoveMem
          (Source,LocalBuffer,Bytes)
DEF FN Global(Source,Bytes) = FN MoveMem
          (LocalBuffer,Source,Bytes)
:
REM ---------------
REM      Main
REM ---------------
:
Xtop = 10 : Ytop = 20
Xbot = 30 : Ybot = 40
:
MODE 7 : REM double high res graphics
CLS
PRINT Xtop,Ytop,Xbot,Ybot:REM before...
FN Local(VARPTR(Ybot),8):REM save data
Xtop = 50 : Ytop = 100 : REM pollute it
Xbot = 90 : Ybot = 170
BOX Xtop,Ytop TO Xbot,Ybot
FN Global(VARPTR(Ybot),8):REM restore it
PRINT Xtop,Ytop,Xbot,Ybot :REM after...
INPUT R$
:
MODE 2
END
```

Part II : Playing with ProDOS

The remainder of this article was inspired by long time, loyal subscriber Jim Shug of Midwest Agri-Business Services. Ol' Jim called me when he noticed that even though he had deleted several large chunks out of a text file one of his programs was maintaining, the actual file size reported in the directory was not getting any smaller.

This occurs because ProDOS cannot really make any assumptions when you don't write up to or past the end of the file. Unlike MS-DOS, ProDOS does not have an end of file "character". Instead, it maintains the file length as part of the information that travels with the file. There is therefore no writing an EOF character to the middle of the file and getting ProDOS to adjust automatically.
That is one of the very few advantages to MS-DOS, I can tell you. And even that advantage is occassionally troublesome. If an EOF gets written mid-file by accident (perhaps an embedded control-character in a file or something) all of the data past the character boils off into the vacuum of space.

There are two solutions to the EOF problem for ProDOS 8.

The first is to avoid the situation altogether ala' AppleWorks™. AppleWorks creates a temporary copy of the file to be saved and then goes back and deletes the original. After that it renames the temporary with the same name as the original.

The problem with this solution is that your users will run out of disk space long before their disks are actually full. If you're working with floppies, as soon as the original AND the temporary file exceed the disk size, you have a problem. AppleWorks 3.0+ solved this by asking users if it could delete the original to get more space. Although the technique works, deleting the original tends to scare the heck out of the non-hacker crowd.

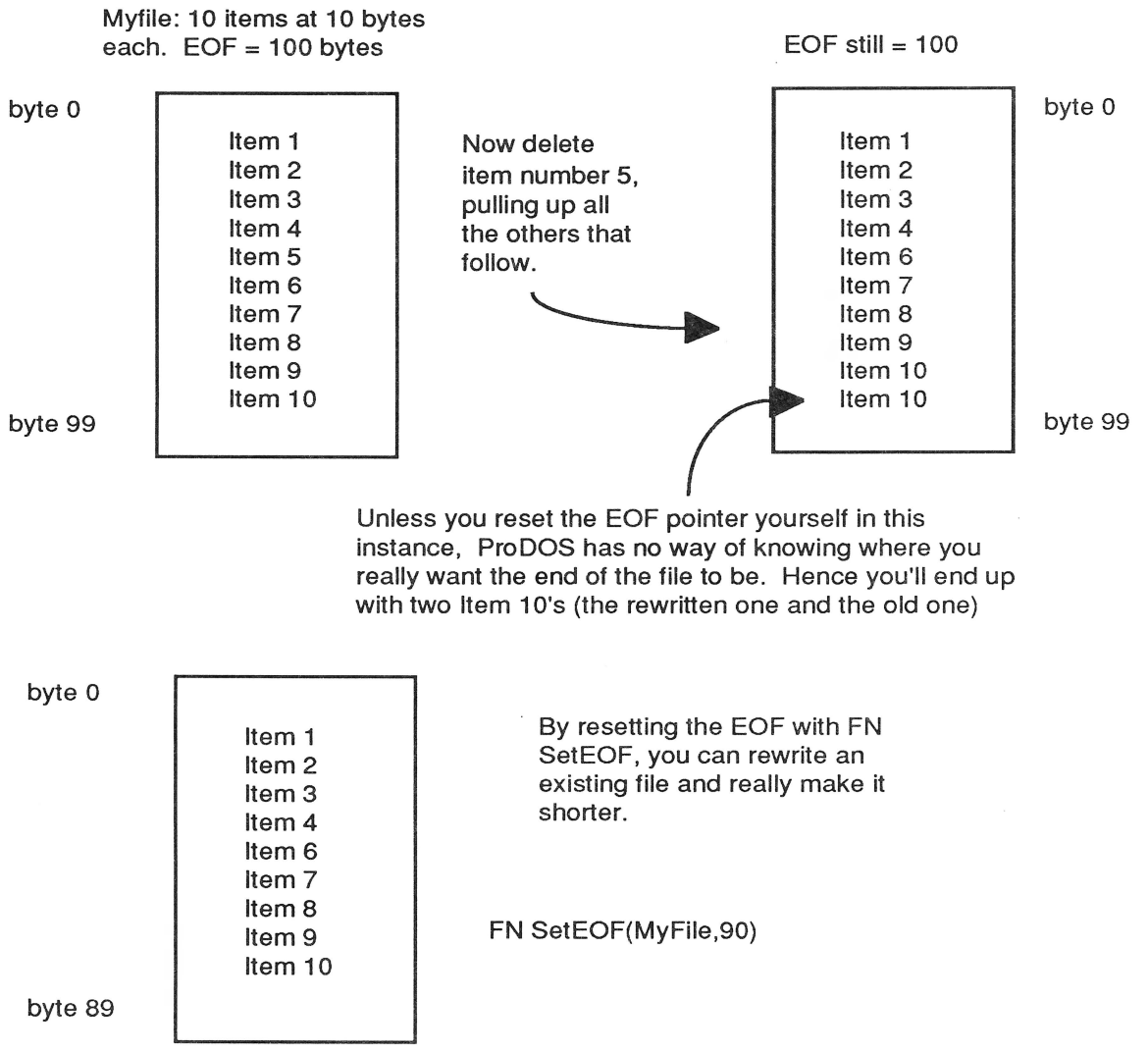
Another solution is the one we'll take here: explicitly set the file length yourself by doing a call to the ProDOS Machine Language Interface. As you'll see, it doesn't really require any knowledge of machine language. And in the process we'll discover a fun little tidbit about mixing Z and MLI calls.

The Tidbit (or TidByte)

Whenever you open a file under ProDOS 8, the DOS assigns a reference number to the open file. This way it can refer to open files by number and not have to dink around with pathnames all the time. This is true whether you open the file with ZBasic's OPEN or do it via assembly language.

To muddy the waters some, the file number that we assign to an open file in ZBasic is most definitely *not* the same as the ProDOS reference number. The ZBasic file number is really for *our* bookkeeping purposes. ProDOS returns a reference number to ZBasic, but ZBasic does not pass it on to us. At least

Figure 1 : Resetting the End of File Pointer to Reflect Deleted Data



not directly.

I must confess that, up until now, whenever I wanted to get the P8 file reference number, I did what everybody else did and wrote a special call to the MLI to OPEN the file and grab the refnum.

Alas, how could I have been so blind? An OPEN call is an OPEN call is an OPEN call. When ZBasic opens a file, it is doing the *exact* same things we are when we code it ourselves. And since the ProDOS ZBasic author, Greg Branche, was so generous with information, we know exactly where ZBasic's own MLI parameter table lives ($1F00).

All we must do is a simple PEEK to the proper spot in the parameter table immediately following the standard ZBasic OPEN statement.

Voile' (say wa-lah'). We have a refnum. A function to do this doesn't even require the LONG variety:

```
REM  This function only works if
REM  called immediately following a
REM  ZBasic OPEN command.

 DEF FN GetRef = PEEK(&1F05)
```

The next step involved in messing with the EOF marker is to find out its current status. As I've discussed before, there are two approaches, one is all-ZBasic and one is more fun.

The all-ZBasic method is to open the file with a record length of 1 byte and then ask how many records there are with an LOF(#) command. It would look like this:

```
OPEN "O",#1,File$,1: REM record length of 1 byte
Len! = LOF(1)      : REM how many records we got?
```

The only trick here is to remember that ProDOS supports file sizes larger than can represented in an integer variable.

A more entertaining approach is to go straight to the MLI. FN GetEOF in the demo below does just that. The only reason it is more entertaining is another little tidbyte: most folks don't know it, but you can save a little time and a little variable space by putting the expression that creates your function's return value right in the END FN = statement. END FN will figure it all out jes' fine, thank you.

The real meat of the demo (if there is any) is to be found in FN SetEOF. This function takes the file ref num as a parameter, as well as the new desired length of the file. Caveat Emptor here, boys and girls (that's Latin for "Hold on to your wallet.") Like the potential boo-boos under MS-DOS, if you set the file length shorter than you mean to, your data will hiss away into the electronic nothingness.

FN SetEOF divides NewLen! by 16^3 (4096) because ProDOS file sizes must be represented by three bytes. The highest-order byte is the number of groups 4096's (like the number of 10's in 100 or something). That's why I indicated this with 16^3 and not 4096. That makes a little sense, doesn't it? You divide by either representation of the number. It's not a time critical operation.

There is one last point of interest in the demo. The main program writes 100 four byte strings to disk ("0123"). I couldn't figure out for the life of me why the file length kept showing up as 500 bytes!

The answer is that the PRINT# statement sends output to disk *exactly* like PRINT does to the screen. In short, there is a carriage return delimeter after each and every string. This fills out each group to five bytes, and makes the entire file 500 bytes long.

Hope this fills your needs Jim, and I hope the rest of you get some use out of it, too.

Listing 2: FN SetEOF and brethren

```
REM ============================
 REM FN SetEOF, FN GetEOF,
 REM
 REM by Ross W. Lambert
 REM Copyright (C) 1990
 REM Ariel Publishing, Inc.
 REM Most Rights Reserved
 REM
 REM I assume that the default
 REM var type is integer
 REM ============================
 :
 :
 DIM 65 Path$:REM   max len plus len byte
 :
 REM Ref is reference number of open file
 :
 REM   This function only works if called
 REM  immediately following a ZBasic OPEN
 REM                                 command
 :
 DEF FN GetRef = PEEK(&1F05)
 :
 :
 LONG FN GetEOF!(Ref)
   POKE &1F00,2 : REM            two parms
   POKE &1F01,Ref: REM  ProDOS ref number
:
   REM                      do GET_EOF call
   MACHLG &A9, &D1, &20, &0865
   :
   REM    put expression in END statement
:
 END FN=PEEK WORD(&1F02)+PEEK(&1F04)*
16.0^3
 :
 :
 LONG FN SetEOF(Ref,NewLen!)
   POKE &1F00,2 : REM            two parms
   POKE &1F01,Ref: REM  ProDOS ref number
   Temp = NewLen!/(16^3)
   POKE &1F04,Temp:REM poke into parm tbl
   REM          subtract to get remainder
```

```
   Temp = NewLen!-Temp
   REM          poke word length remainder
   POKE WORD &1F02,Temp
   REM                        do SET_EOF call
   MACHLG &A9, &D0, &20, &0865
 END FN = ERROR
 :
 REM ------------------
 REM       Main
 REM ------------------
 :
 MODE 2
 :
 REM            Create a dummy file at the
current prefix
 :
 KILL "DUMMY"
 IF ERROR <> 0 THEN ERROR = 0
 OPEN "O",1,"DUMMY"
 IF ERROR <> 0 THEN STOP
 Ref = FN GetRef : REM    get ProDOS
internal reference #
 :
 FOR X = 0 TO 99
   PRINT #1,"0123" : REM  implicit car-
riage return here!
 NEXT
 :
 PRINT "The current file length is: ";FN
GetEOF!(Ref);" bytes"
 CLOSE
 :
 REM   Go write some bytes in the middle
 :
 OPEN "O",1,"DUMMY",1 : REM record len=1
 IF ERROR <> 0 THEN STOP
 Ref = FN GetRef
 RECORD#1,20:REM        write bytes 20-24
 PRINT#1,"56789"
 FN SetEOF(Ref,25):REM     set new length
 PRINT "The new length is: ";FN
GetEOF!(Ref)
 CLOSE
 :
 END
```

LRO Computer Sales

665 West Jackson Street, Woodstock, IL 60098

Mon-Fri, 9-6 CST **(800) 869-9152 (815) 338-8685** **Sat 12-5 CST**

Memory

GS-4 Memory Board

0k	$49	1 Meg	$99
2 Meg	$166	4 Meg	$289

Chinook RAM 4000

0k	$75	1 Meg	$139
2 Meg	$199	4 Meg	$319

GS-Sauce SIMM Board

0k	$89	1 Meg	$161
2 Meg	$230	4 Meg	$369

GS Ram +

1 Meg	$212	2 Meg	$279
3 Meg	$344	4 Meg	$411
5 Meg	$475	6 Meg	$535

Checkmate MemorySaver $119

All memory is new and has a **5 year warranty.**

Apple 1 Meg 80ns exp. set	$67
SIMM expansion set	$69
Apple 256k 120ns exp. set	$18
Apple 256k X 4 exp. set	$19

Accesories for GS

Transwarp GS 7 Mhz	$279
Sonic Blaster	$96
VisionaryGS Digitizer	$279
RamFast 256k DMA SCSI	$197
Sound System II speakers	$99
System Saver GS	$69
Conserver GS	$89
A+ Optical Mouse ADB	$87
Cordless Mouse ADB	$109

GS Hardware

* Apple IIGS ROM 01 CPU	$649
Apple IIGS 1 Meg CPU, keyboard and mouse	$819
Apple Color RGB Monitor	$447
Apple IW II w/32k buffer	$449
Magnavox RGB Monitor	$319
Fortris ImageWriter compatible printer	$229
HP DeskJet+ 300 DPI!	**$599**
Æ 3.5" Drive upgradable from 800k to1.44Meg	$219
AMR 3.5" Drive	$183
AMR 5.25" Drive	$149

Software

Utilities

Copy II Plus v. 9.0	$25
Print Shop GS	$27
ProSel 8/16	$66
Programmers's Online Companion	$37.50

Vitesse Salvation Series:

Guardian— HD Backup	$29
Renaissance— Optimizer	$29
Exorciser— Virus Detector	$26

Graphic Disk Labeler v.2.0
Print Color Disk Labels on IW II in 320 and 640 modes!
$24.50

Business

AppleWorks GS	$212
Manzanita Businessworks	$294

Education

Designasaurus GS	$33
Geometry GS	$56
Talking Once Upon a Time	$34
StudyMate- Grade Booster	$33

GS Numerics
A complete math program for high school, college students and professionals
$104

Zip GS 8 Mhz $269

Entertainment

FutureShock v.2.0	**$54**
Heatwave Offshore Racing	$37
Test Drive II: The Duel	$34
Grand Prix Circuit	$36
Blue Angels Flight Sim.	$37
Third Courier	$37
Jam Session	$32.25
Task Force	$29
California Games	**$14.50**
Qix	$25
Rastan	$25
Arkanoid I or II	$25
Chessmaster 2100	$37
Tunnels of Armageddon	$32

GS Starter System

- Apple IIGS 1 Meg CPU, keyboard and mouse
- Magnavox RGB Monitor
- Fortris ImageWriter compatible printer
- AMR 3.5" Drive
- Mouse pad
- Box of 10 Maxell 3.5" Disks

$1599

GS Power System

- Apple IIgs 1 Meg CPU, keyboard and mouse
- Apple Color RGB Monitor
- Apple ImageWriter II with 32k buffer
- Apple High Speed DMA SCSI
- AMR 40 Meg GS Partener HD
- Chinook RAM 4000 w/ 2 Meg
- AMR 3.5" Drive
- Mouse pad
- Box of 10 Maxell 3.5" Disks

$2959

Modems

USR 14.4 kbs Courier HST	$589
Cardinal 2400 baud	$109
Supra 2400 baud	$109
Prometheus Promodem internal 2400G	$144

Hard Drives

Chinook CT100 16k cache			$780
UniStore 80 Meg HS HD			$529
UniStore 60 Meg HS HD			$474
AMR GS Partner (0 Footprint)			
40 Meg	$420	60 Meg	$640
80 Meg	$700	100 Meg	$876
AMR 45 Removable HD			$769
CMS 60 Meg HD			$539

Apple DMA SCSI w/ purchase of HD: $96 Without: $101

All HDs come formatted w/ GSOS or Mac system software, and 5-10 megs of PD, Share/Freeware,NDAs, CDAs, and INITs.

InnerExpress $85

Prices subject to change without notice. Returns within 15 days with no restocking fee. IL residents add 6.5%. FAX orders and receive 2nd day air upgrade! (815) 338-8597

Programming the Extended Keyboard

Getting Over Extended

By Dr. Doni G. Grande

Many of us lead a dark, dual existence. By day, we are tied to some three-letter computer at work, slaving away at a keyboard with multiple function keys, and by night we are released and able to soar with our fantastic IIgs's, with their less-endowed keyboards. But the curse of this lifestyle is that we are forever having to relearn the keyboard layout when we switch between machines.

This is where an extended keyboard comes to the rescue. The key layout on the Apple extended keyboard is very similar to the "enhanced keyboard" many of us use on those other computers. There are fifteen function keys (called F-keys and labeled F1-F15) across the top of the keyboard. The cursor movement keys (in an inverted-T configuration) and an editing keypad (with keys labeled help, home, page up, delete, end, and page down) are located between the main alphanumeric keys and the numeric keypad. There are even three lights (num lock, caps lock, and scroll lock) located above the keypad.

Where'd it go, George, where did it go?

How do you know when those extra keys are pressed? Well, the extra keys act like they are on a big extended keypad. Table 1 lists the key equivalents for these keys. In case you are wondering where the "keypad bit" is located, its kept in the key modifier register located at $E0/C025 (or PEEK(49189) from Applesoft) as shown in Table 2.

Listing 1 is a short Applesoft program that GETs a keyboard character and the key modifier register and prints which keys were hit. The key modifier register (read into KEYMOD) is parsed by checking for the higher bits first. If set, the appropriate modifier is added to the description string and the bit is subtracted from KEYMOD. This is continued until no bits are left set in KEYMOD, and then the string is printed out. Note that control characters are printed with a caret preceding the letter; thus, control-E prints as ^E. It is interesting to notice that some keys return control characters (RETURN, ESC, TAB, and the arrow keys) without setting the CTRL bit since control is not actually held down to generate these characters (but the keyboard WILL correctly report CTRL ^M if you hold down CONTROL and hit RETURN, as well as KP ^M if you press ENTER on the keypad!).

Unfortunately, most Apple IIgs programs don't check for the keypad bit, so they just appear to return the characters in Table 1 (F1 gives you a "z", F2 an "x", etc.). This isn't very useful. Perhaps as more people get the extended keyboard, more programs will be adapted to them.

"...a Permanent Initialization File (PIF) can be written that will be installed when the IIgs boots and will set the keyboard lights when any desired modifier keys are down."

UltraExtended

Some programs can be made to recognize the function keys, however. For example, Beagle Bros' Time-Out UltraMacros 3.0 will accept extended keyboard keys to trigger macros, so you can setup the function

keys in AppleWorks. The only problem with this approach is that each key is mapped into its equivalent solid-Apple (SA) key sequence, and most of the predefined UltraMacros already use the Fkey equivalents. For example, HOME maps into SA-s, which is already used for the "Save File and Remove" macro. After discussing this on the Beagle Bros Support BBS, Mark Munz sent me a patch that would let the F-keys map into the Both-Apple (BA) macros, which are much less used. This patch and the standard set of macros I use are in listing 2. *(Editor: Please note that our beta tester had trouble with the UltraMacros part of this presentation. I elected to print them anyway due to the very broad interest in the subject. Caveat Emptor.)*

Turnin' on the lights

Finally, there's the three lights on the keyboard: NUMLOCK, CAPS LOCK, and SCROLL LOCK? It seems that Apple thought about how to handle them, albeit in hindsight. The ROM 3 machines will follow the status of the CAPS LOCK key with the CAPS LOCK light, but the earlier ROM 0 and ROM 1 machines don't. Let's remedy that!

Figuring out exactly what to do takes a little bit of digging. The Apple IIGS Toolbox Reference Volume One documents the Apple Desktop Bus toolset. There is a SendInfo routine documented to send data to ADB devices, on pages 3-19 through 3-22. However, how this relates to the extended keyboard is not documented. By disassembling a program by Tracy Valleau that set the CAPS LOCK light and a little playing around with some values, the following parameter list was discovered:

ADB SendInfo parameters to set keyboard lights:

```
2 byte count for ADB call
4 byte pointer to ADB data
2 byte $00A2 - send two bytes to
      ADB device 2, register 1
```

The ADB data should be of the form (in binary):

```
1111 1xyz 1111 1111

where x = 1 for numlock off,0 for on
      y = 1 for capslock off,0 for on
      z = 1 for scroll lock off,0 for on
```

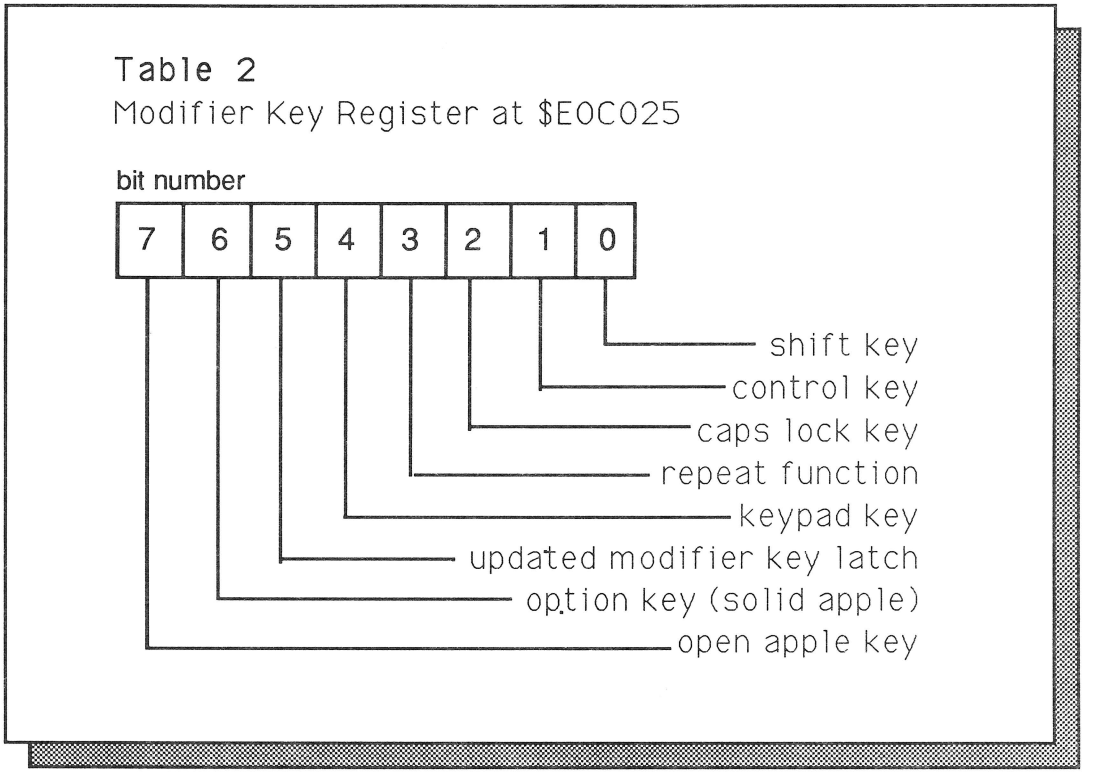
Knowing this, a Permanent Initialization File (PIF) can be written that will be installed when the IIgs boots and will set the keyboard lights when any desired modifier keys are down. The PIF source is shown in listing 3, with the "make" file in listing 4.

Table 1
Extended Keyboard Function Key Equivalents
All values are returned with the keypad bit set.

F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
z	x	c	v	'	a	b	d	e	m	g	o	i	k	q

page +

help	home	up	del	end	down
r	s	t	u	w	y



This PIF is the one originally written by Tracy Valleau modified to allow any of the keyboard lights to follow any of the modifier keys. As the code now stands, it merely follows the status of the CAPS LOCK key with the CAPS LOCK light. To change which modifier keys control which lights, change the constants at the end of the program to reflect your desires. For example, changing the constants to:

```
numlight  dc i2'apple+option'
caplight  dc i2'capslock'
scrlight  dc i2'ctrldwn+keypad'
allmods   dc i1'apple+option+capslock+
               ctrldwn+keypad'
```

...would make the NUM LOCK light follow the status of either Apple key, the CAPS LOCK light follow the CAPS LOCK key, and the SCROLL LOCK light follow the CONTROL and KEYPAD keys. The allmods constant must contain all the modifiers you are interested in; it makes the code quicker in deciding if anything needs to be done when a modifier key is down. *(A desktop program that allows you to customize the PIF is available on the 8/16 monthly disk. -ed)*

Listing 1
Applesoft Modifier Key Register Display

```
 5  REM  Program to show use of Modifier
Key register ($C025)
 10  GET A$:A =  ASC (A$): IF A < 32 THEN
A$ = "^" +  CHR$ (A + 64): REM Convert
control chars
 20 KEYMOD =  PEEK (49189): REM = $C025
 30 KM$ = "": IF KEYMOD > 127 THEN KEYMOD
= KEYMOD - 128:KM$ = "OA "
 40  IF KEYMOD > 63 THEN KEYMOD = KEYMOD
- 64:KM$ = KM$ + "CA "
 50  IF KEYMOD > 31 THEN KEYMOD = KEYMOD
- 32:KM$ = KM$ + "KEYMOD "
 60  IF KEYMOD > 15 THEN KEYMOD = KEYMOD
- 16:KM$ = KM$ + "KP "
 70  IF KEYMOD > 7 THEN KEYMOD = KEYMOD -
8:KM$ = KM$ + "RPT "
 80  IF KEYMOD > 3 THEN KEYMOD = KEYMOD -
4:KM$ = KM$ + "CAPSLOCK "
 90  IF KEYMOD > 1 THEN KEYMOD = KEYMOD -
2:KM$ = KM$ + "CTRL "
 100  IF KEYMOD > 0 THEN KM$ = KM$ +
```

```
"SHIFT "
 110  PRINT "Key hit: ";KM$;A$
 120  GOTO 10
```

Listing 2
Ultramacros Macro to Enable Extended Keyboard Fkeys

(A reminder: some folks have no problem with these, others find that it locks up their computer or crashes it. This may be due to inteference with other INITs or the effects of cosmic rays. - editor)

Here is a fix for the extended keyboard. All the function keys are turned into ba- macros instead of sa- macros. Thanks to Mark Munz for these!

Note: these patches use memory from $30E-318. Anything that writes over this area will crash AppleWorks! The STARTUP macro appears to trash this memory, so you can't have this patch automatically installed when AppleWorks boots.

```
<ba-ctrl-e>:<all:A=782:poke A+0,32 : poke
a+1,3 : poke a+2,181 : poke a+3,208 :
poke a+4,3 : poke a+5,76 : poke a+6,13 :
poke a+7,182 : poke a+8,76 : poke a+9,20
: poke a+10,182 : pokeword 4503,$30E: msg
'BA extended keys'>!

<ba-ctrl-n>:<all:pokeword 4503,$B60D: msg
'SA extended keys'>!

<ba-S>:<awp:oa-,>! Home key goes to be-
ginning of line.
<ba-w>:<awp:oa-.>! End goes to the end of
line.
<ba-t>:<awp:oa-up>! Page up goes up one
page
<ba-y>:<awp:oa-down>! Page down goes down
one page
<ba-r>:<all:oa-?>! Help shows help screen
<ba-u>:<all:oa-del>! Deletes character
under the cursor

<ba-z>:<all:sa-u>! F1 - UNDO - Undo last
delete - paste from clipboard
<ba-x>:<all: oa-m>T! F2 - CUT - Move data
to clipboard
<ba-c>:<all: oa-c>T! F3 - COPY - Copy
data to clipboard
<ba-v>:<all: sa-u>! F4 - PASTE - copy
from clipboard
```

Listing 3
LIGHTS PIF to Enable Extended Keyboard Lights

```
;
; LIGHTS V1.0 - 26 Jun 1990
;
;  By Doni G. Grande
;
;  A PIF that enables the lights on an
;  extended keyboard.  One or more of the
;  modifier keys may be linked to any of
the
;  three lights (numlock, capslock, or
scroll
;  lock). Use the CONFIG.LIGHTS program
to
;  configure which keys control the
lights.
;
; Based on the MYLIGHTS program by Tracy
; Valleau.  This driver was created from
; a disassembly of MYLIGHTS (made with
;ORCA/Disassembler) which was modified to
; add support for lights and keys other
; than capslock.
;
; Copyright (c) 1990 Ariel Publishing and
; Doni G. Grande. Some rights reserved.

; Get our macros

         mcopy lights.mac

;
; Predefined labels:

OS_KIND  gequ  $E100BC
PRODOS8  gequ  $BF00
PREFLAG  gequ  $BF9A
INCBUSYFLG gequ $E10064
DECBUSYFLG gequ $E10068
KEYMODREG gequ $C025
MTSptr   gequ  $00

;
; Define bits for modifier key register
;
```

```
shiftdwn gequ  $01          Shift key down
ctrldwn  gequ  $02        Control key down
capslock gequ  $04      Caps Lock key down
anydown  gequ  $08        Some key is down
keypad   gequ  $10         Keypad key down
option   gequ  $40         Option key down
apple    gequ  $80          Apple key down

lights   start

;  Start w/some housekeeping
;  Get direct page and start some tools
;  we need

         phk
         plb
         rep   #$30
         longa on
         longi on
         _TLStartUp    Start Tool locator
         pha
         _MMStartUp  Start memory manager
         _MTStartUp    Start misc toolset
;
; We need a ptr in zero page, so preserve
; the original contents.
;
         lda   MTSptr+2   save zero page
         pha                    locations
         lda   MTSptr
         pha
;
; Have to patch the Misc Toolset's reset
; rtn to reinstall our heartbeat task.
;
         pea   $0000         Reserve space
         pea   $0000            for result
         pea   $0000       Get system tool
         pea   $0003     Want Misc Toolset
         _GetTSPtr                Get ptr

         pla               Move pointer to
         sta   MTSptr           zero page
         pla
         sta   MTSptr+2
;
;The reset routine is number $0014. Load
;the original value and place in our code
;so we can call when needed.  Then patch
;the Misc Toolset reset routine to point
;to our routine.
;
         ldy   #$0014    loword of addr-1
         lda   [MTSptr],y of Misc Tool
                              Reset

         inc   a     Point to addr of MTS
         sta   oMTSrst  MTS reset routine
         lda   #ResetFix
         dec   a
         sta   [MTSptr],y

         ldy   #$0016  Get hibyte addr-1
         sep   #$20          of MTS reset
         longa off                routine
         lda   [MTSptr],y
         sta   oMTSrsthi
         lda   #^ResetFix
         sta   [MTSptr],y
;
; Restore zero page to way we found it.
;
         rep   #$20
         longa on
         pla             Restore zero page
         sta   MTSptr             location
         pla
         sta   $02
;
; Install our heartbeat routine
;
         jsl   >InstHB
;
; Shutdown the tools we started
;
         _MTShutDown
         _MMShutDown
         _TLShutDown

         rtl
;
; This is reset routine patch.  The first
; instruction will be patched to point to
; original Misc Toolset reset rtn. After
; executing the original routine, kybd
; light heartbeat task is re-installed.
;
ResetFix dc    h'22'           JSL opcode
oMTSrst  dc    h'0000'  Orig Misc Toolset
oMTSrsthi dc   h'00'        Reset address
InstHB   pea   HBtskptr|-$10  Install our
         pea   HBtskptr    heartbeat task
         _SetHeartBeat
```

```
         pea   $0002      Enable VBL ints
         _IntSource           (vblEnable)
         rtl

; This is actual heartbeat task routine.

HBtskptr dc    h'00000000'   Heartbeat
task header
HBCount  dc    i2'$000A'     Six inter-
rupts/sec
HBSig    dc    h'5AA5'       Heartbeat
signature
         php
         rep   #$30          Switch to
native mode
         longa on
         longi on
         phk
         plb
;
; Check to see if the current operating
; system is busy. If so, shorten the wait
; time to five ticks & return; otherwise
; go check the keyboard modifiers.
;
         lda   >$E100FF
         and   #$00FF
         bne   WaitFive
         lda   >OS_KIND indicates current
         and   #$00FF          running O/S
         bne   doit
         lda   >PRODOS8
         and   #$00FF
         cmp   #$004C
         bne   WaitTen
         lda   >PREFLAG        prefix flag
         bmi   WaitFive
doit     jsl   >INCBUSYFLG  inc busy flag
         jsr   ChkCAPS
         jsl   >DECBUSYFLG  dec busy flag
WaitTen  lda   #$000A
         bra   SetHBC
WaitFive lda   #$0005
SetHBC   sta   HBCount
         plp
         rtl

; This rtn does all the work of checking
; the kbd modifiers & setting the lights.

ChkCAPS  sep   #$30
         longa off
         longi off
         lda   >KEYMODREG Get key mod reg
         and   allmods Strip bits we want
         sta   CAPstatus     Save current
         rep   #$30                status
         longa on
         longi on
         lda   CAPstatus Compare caps now
         eor   PrevCAPS    with previous.
         bne   NotEqual     Go if changed
         rts             Otherwise return

NotEqual anop           Update the lights
         stz   ADBdata Zero ADB data byte

         lda   CAPstatus Get light status
         sta   PrevCAPS Save 4 next pass
         pha
         and   numlight Check numlight on
         beq   nonum             Go if not
         sec             Roll bit into the
         rol   ADBdata      ADB data byte
nonum    pla              Get light status
         pha
         clc
         and   caplight              Check
capslock on
         beq   nocap             Go if not
         sec             Roll bit into the
nocap    rol   ADBdata      ADB data byte
         pla
         clc
         and   scrlight Check scroll lock
         beq   noscrl            Go if off
         sec             Roll bit into the
noscrl   rol   ADBdata      ADB data byte

         lda   ADBdata   Get ADB data val
         eor   #$FFFF     Invert all bits
;
;Here is the rtn that does all the work.
;It uses SendInfo ADB Toolset call to set
;the lights.  This is documented in the
;Toolbox Reference Vol 1, but the parm
;used here is not mentioned.From what
;we've been able to figure out, the
;parameters are:
;
;    2 byte count for ADB call
;    4 byte pointer to ADB data
;    2 byte $00A2 - send two bytes to ADB
;                       device 2 register 1
```

```
;
; The ADB data should be of the form
;  (in binary):
;
;    1111 1xyz 1111 1111
;
;   where x = 1 for numlock off, 0 for on
;         y = 1 for capslock off, 0 for on
;     z = 1 for scroll lock off, 0 for on
;

SendData anop              Send data to kbd
         xba               Swap hi/lo bytes
         sta   ADBdata       Save in buffer
         pea   $0002    Send ADB two bytes
         lda   ADBdataptr+2    Ptr to data
buffer
         pha
         lda   ADBdataptr
         pha
         pea   $00A2       Send to register
1
         _SendInfo
         rts

ADBdata  dc    i2'$0000' Data for ADB kbd
ADBdataptr dc  a4'ADBdata'    Ptr to above
PrevCAPS dc    i2'$0000'Previous CAPS stat
CAPstatus dc   i2'$0000' Present CAPS stat
;
;Marker for data area.  This is used by
;configure program (on 8/16 monthly disk)
;to find the data area in the disk file.
;
         dc    c'data'
;
; The following determines which lights
; correspond to which keys.
;

numlight dc    i2'$0000'
caplight dc    i2'capslock'
scrlight dc    i2'$0000'
allmods  dc    i1'capslock'
; All mod keys we
; are interested in
         end
```

Listing 4
Makefile for LIGHTS PIF

```
asml lights.asm keep=lights
erase lights.root
filetype lights str
```

New Concepts Software

DreamGraphix: A full feature paint program and more!

- 16, 256, and 3200 color and palette control with ability to copy, spread, and swap colors and palettes
- Draw filled or outlined circles, ovals, squares, rectangles, polygons, arcs, lines, and any other shapes
- Mask color, spray paint, fill region (solid, pattern, and brush), freehand, zoom (with and without grid)
- Use anything as a brush with ability to stretch, halve, double, rotate, flip, bend, shear and mirror
- Smear, shade, blend, and smooth commands • Use all available fonts • Hard drive instalable
- Load and save pictures on all available and custom 3200 formats on the GS • Load and save palettes
- **User expandable through extended commands**

Inner*Express*: Speeds up Falcons, Inner and OverDrives 200%—300%

InnerDrive without Inner*Express*		InnerDrive with Inner*Express*		ID with Inner*Express* and TWGS	
Interleave:	10:1	Interleave:	6:1	Interleave:	5:1
Block access/second:	81	Block access/second:	225	Block access/second:	284

FutureShock V 2.0: Entertainment system with FuturePad controller

FutureShock is a revolutionary game with its own interface controller- FuturePad. To manipulate the objects in the game, you move your hand over the pad! These games (DeathSquash, Zertotron Racer, and Metal Heads) can played by two players through modem. It comes with its own editor for making levels for the game!

Limited Time Offers Through September 15th: *DreamGraphix $42 Inner*Express* *$95* FutureShock $59

Free 2nd day shipping with any order!

Dealer Inquires Welcomed **New Concepts Software:** 1-800-487-8684

Classic Apple Hacking

Generic StartUp 8

by Jerry Kindall

Carl Hilton asked me on GEnie why the Sourceror's Apprentice had once run a generic GS/OS startup and shutdown routine, but never an 8-bit version of the same song and dance. It sounded like a good idea to me. It's not exactly crystal clear what one must do at the beginning of a ProDOS 8 SYS application. I took the front end I'd written for MicroDot and extended it to make it as generic as possible. The result was Listing 1.

I designed the program so that it can be included easily in your assembly by using the Merlin PUT directive. Here's how a SYS application might start:

```
1   * The world's greatest program
2   * by Ima Bozo
3
4     org   $2000  ;SYS files ORG @ $2000
5     typ   $FF    ;SYS files type $FF
6     dsk   GREATPROG   ;assemble to disk
7
8     put   generic ;include generic startup rtn
9
10  * Program begins here...
```

I tried to minimize the impact the Generic Startup Routine would have on your program's own labels. Many internal labels are local labels, rather than global labels. The only global labels in the routine (besides the equates, some of which you may find useful anyway) are:

strtpath: the startup path passed to this application by the launcher
start: the starting address of the generic startup routine
progpath: the program's complete pathname, or null if indeterminable
main: the starting address of the program code you write

I won't go into great detail describing the program's inner workings. It's well commented, and besides, writing the program was mainly a matter of looking through various manuals and figuring out what most SYS programs need to do, then writing the code. I've pointed out (in the comments) places where the routine can be modified to better suit your needs.

The routine isn't particularly tricky, but I sure am glad I'll never have to worry about it again. I hope it saves you a headache or two, as well. Be on the lookout for a generic shutdown routine next month!

Listing 1: The Generic 8-bit Startup Routine

```
1
2     * Generic 8-bit Startup Routine
3     * by Jerry Kindall - 8/16
4
5              lst   off
6
7     csw     =     $36         ;output hook
8     ksw     =     $38          ;input hook
9     resvec  =     $3F2       ;reset vector
10    settxt  =     $FB39          ;set text
11    home    =     $FC58      ;clear screen
12    setnorm =     $FE84   ;select norm vid
13    setkbd  =     $FE89       ;select IN#0
14    setvid  =     $FE93       ;select PR#0
15    coldst  =     $E000   ;coldstart BASIC
16
17    fretop  =     $6F  ;beginning of strs
18    memsiz  =     $73      ;end of strings
19
20    GETINFO =       $C4 ;ProDOS MLI codes
21    ONLINE  =       $C5
22    SETPFX  =       $C6
23    GETPFX  =       $C7
24    CLOSE   =      $CC
25
26    mli     =      $BF00  ;ProDOS entry pt
27    devnum  =      $BF30;last disk accessd
28    bitmap  =      $BF58;memory protection
29    level   =     $BF94 ;file access level
30    machid  =      $BF98  ;machine id byte
```

```
31   pfxflag=     $BF9A;prefx active flag
32
33   * Hardware & Firmware:
34
35   kbd    =     $C000    ;read keypress
36   strobe =     $C010   ;clear keypress
37   page1  =     $C054    ;select page 1
38   ent80  =     $C300;80column entry pt
39
40   fn1    =     $280        ;filename 1
41   fn2    =     $2C0        ;filename 2
42
43           jmp   start;jmp over startup buffer
44
45 * Startup buffer
46 * This code can be omitted if you
47 * don't want the ability to pass a
48 * string to your program. Otherwise
49 * it's kind of nifty to have.
50             hex   EEEE41 ;startup buffer ID
bytes
51
52   strtpath str   'STARTUP'  ;omit this line
for null
53                         ; startup path
54
55             ds    65-*+strtpath ;filler
56
57   * Close all open files and init stack
pointer
58
59   start     lda   #0 ;level set to 0 for all
files
60             sta   level
61             jsr   mli
62             dfb   CLOSE
63             dw    :pclose
64             ldx   #$FF  ;stack pointer should
be set to
65             txs     ; top of stack page
66
67*Check for and deactivate 80-columns
68*If you want instead to activate 80columns if
69*available, use lda #$19 instead of lda #$15.
70*You could require 80 columns by branching to
71*error handling routine instead of :inittxt.
72
73           lda   machid  ;get machine id
74           and   #$02       ;check bit 1
75           beq   :inittxt   ;no 80-col card
installed
76           lda   #$15       ;turn it off
77           jsr   ent80      ;$C300 entry
78
79*Set up display
80*This is executed on re-entry from Applesoft
81*(when Applesoft calls RDKEY to get keypress)
82
83   :inittxt jsr   setkbd          ;IN#0
84             jsr   setvid          ;PR#0
85             jsr   setnorm ;normal video
86             jsr   settxt     ;text mode
87             bit   page1 ;display page 1
88             jsr   home    ;clear screen
89
90   * Get BASIC going If you will not be using
91   * Applesoft BASIC routines, such as FP math,
92   * you may leave out this section
93
94             lda   #:reentry  ;this is our
return ticket
95             sta   ksw
96             lda   #/:reentry
97             sta   ksw+1
98             lda   #:nullout  ;eat all output
chars
99             sta   csw
100            lda   #/:nullout
101            sta   csw+1
102            jmp   coldst ;coldstart Applesoft
103
104  :reentry lda   #$BF;protect ProDOS global pg
105            sta   memsiz+1  ;from BASIC strings
106            sta   fretop+1
107            lda   #0
108            sta   memsiz
109            sta   fretop
110
111  * Set memory protection
112  * Clears out ProDOS bitmap & reserves pages
113  * 0,1,4-7, $BF. Add other protection here if
114  * necessary.
115
116            lda   #0
117            ldx   #$17
118  :memloop sta   bitmap,x  ;all mem accessible
119            dex
120            bne   :memloop
121            lda   #%11001111 ;except 0,1,4-7
122            sta   bitmap
123            lda   #%00000001 ;$BF
124            sta   bitmap+$17
125
126 * Avoid null prefix
127 * If null prefix found, set prefix to name of
128 * volume last accessed.
129
130  :prefix  lda   pfxflag
131           bne   :getpath   ;not null
132           lda   devnum     ;get current disk
133           sta   :ponline+1
134           jsr   mli
135           dfb   ONLINE
136           dw    :ponline
```

```
137           lda   fn1+1
138           and   #$0F        ;if length zero
139           tax
140           lda   #'/'        ;put beg slash
141           sta   fn1+1
142           inx
143           stx   fn1         ;save new length
144           jsr   mli         ;and set prefix
145           dfb   SETPFX
146           dw    :psetpfx
147
148*Get application pathname
149*If you do not need to know the name of your
150*app, or the directory it's in (possibly
151*different from launch prefix), skip this.
152*When this routine has completed, the memory
153*area "progpath" will contain full pathname
154*of application program.  If the information
155*was unavailable (or pathname was too long)
156*the str will be null. You may wish to strip
157*the last name in path in order to get the
158*application directory, then set prefix to
159*that directory.  This allows your program
160*to find its files even if the prefix wasn't
161*set to application's directory.  It also
162*assumes that the launching program followed
163*ProDOS convention of placing application
164*pathname at $280.
165
166  :getpath ldy   #0
167           ldx   #0
168           lda   fn1+1;check slash at strt
169           cmp   #'/'
170           beq   :fulpath  ;full path speci-
fied by launcher
171           jsr   mli
172           dfb   GETPFX
173           dw    :pgetpfx
174           ldx   progpath
175  :fulpath lda   fn1+1,y
176           sta   progpath+1,x
177           iny
178           inx
179           cpx   #64     ;pathname overflow?
180           beq   :chkpath                ;yes
181           cpy   fn1 ;no, done copying path?
182           bne   :fulpath       ;no, do more
183
184  :chkpath stx   progpath
185           jsr   mli   ;get info on pathname
186           dfb   GETINFO
187           dw    :pgetinf
188           bcc   :reset           ;no errors
189           lda   #0;otherwise we've an error
190           sta   progpath       ; make null
191
192 * Set up reset vector
193 * This code makes program restart if Reset
194 * is pressed. Probably you need to change
195 * where program goes when Reset is pressed.
196
197  :reset   lda   #main
198           sta   resvec
199           lda   #/main
200           sta   resvec+1
201           eor   #$A5 ;set funny complement to
202       sta   resvec+2;avoid rebooting on reset
203
204  * Clear keyboard buffer
205  * Clears all buffered keystrokes on IIgs
206
207  :clrkey  bit   kbd
208           bit   strobe
209           bmi   :clrkey
210
211 * Begin program execution
212
213           jmp   main
214
215 * Fake RTS to eat output during BASIC startup
216
217  :nullout rts
218
219  * MLI parmlists and data areas
220
221  :pgetinf dfb   $0A         ;GET_INFO pathname
222           dw    progpath             ;pathname
223           ds    15  ;we don't care about rest
224
225  :pgetpfx dfb   $01       ;GET_PREFIX pathname
226           dw    progpath             ;pathname
227
228  :psetpfx dfb   $01       ;SET_PREFIX pathname
229           dw    fn2                  ;pathname
230
231  :ponline dfb   $02           ;ONLINE parmlist
232           dfb   $00               ;unit number
233           dw    fn2+1             ;data buffer
224
235  :pclose  dfb   $01            ;CLOSE parmlist
236           dfb   $00       ;ref num (all files)
237
238  progpath ds    65       ;application pathname
239                                ;null if unknown
240                ;unneeded if you omit the code
241                        ; which initializes it
242
243  main                  ;yer program goes here
244           lst   off
```

GENESYS

GS Sauce RAM Card

Now available and shipping!

Genesys™...the premier resource creation, editing, and source code generation tool for the Apple II GS.

Genesys is the first Apple IIGS CASE tool of its kind with an open-ended architecture, allowing for support of new resource types as Apple Computer releases them by simply copying additional Genesys Editors to a folder. Experienced programmers will appreciate the ability to create their own style of Genesys Editors, useful for private resource creation and maintenance. And Genesys generates fully commented source code for ANY language supporting System 5.0. Using the Genesys Source Code Generation Langugage (SCGL), the Genesys user can tailor the source code generated to their individual tastes, and also have the ability to generate source code for new languages, existing or not.

Genesys allows creation and editing of resources using a WYSIWYG environment. Easily create and edit windows, dialogs, menu bars, menus menu items, strings of all types, all the new system 5.0 controls, icons, cursors, alerts, and much more without typing, compiling, or linking one single line of code.

The items created with Genesys can be saved as a resource fork or turned into source code for just about any language. Genesys even allows you to edit an existing program that makes use of resources.

Genesys is guaranteed to cut weeks, even months, off program development and maintenance. Since the interface is attached to the program, additions and modifications take an instant effect.

Budding programmers will appreciate the ability to generate source code in a variety of different languages, gaining an insight into resources and programming in general. Non-programmers can use Genesys to tailor programs that make use of resources. Renaming menus and menu items, adding keyboard equivalents to menus and controls, changing the shape and color of windows and controls, and more. The possibilities are almost limitless!

Genesys is an indispensable tool for the programmer and non-programmer alike!

Retail Price: $150.00

SSSi is pleased to announce that we will be carrying the GS Sauce memory card by Harris Laboratories. This card offers several unique features to Apple //gs owners:

- Made in USA
- Limited Lifetime Warranty
- 100% DMA compatable
- 100% GS/OS 5.0 and ProDOS 8 & 16 compatable
- Installs in less than 15 seconds!
- Low-power CMOS chips
- Uses "snap-in" SIMMs modules - the same ones used on the Macintosh
- Recycle your Macintosh SIMMs modules with GS Sauce.
- Expandable from 256K to 4 Meg of extra DRAM

This card is 100% compatable with all GS software and GS operating systems. It is 100% tested before shipping and has a lifetime warranty. The CMOS technology means that it consumes less power and produces less heat thus making it easier on your //gs power supply. There are no jumpers, just simple to use switches to set the memory configuration. One step installation takes less than 15 seconds.

Memory configurations:

Apple //gs model	add these:	total GS RAM
256K (ROM 1)	(1) 256K SIMM	512K
	(2) 256K SIMMs	768K
	(4) 256K SIMMs	1.25 Meg
	(1) 1 Meg SIMM	1.25 Meg
	(2) 1 Meg SIMMs	2.25 Meg
	(4) 1 Meg SIMMs	4.25 Meg
1 Meg (ROM 3)	(1) 256K SIMM	1.25 Meg
	(2) 256K SIMMs	1.50 Meg
	(4) 256K SIMMs	1.78 Meg
	(1) 1 Meg SIMM	2.0 Meg
	(2) 1 Meg SIMMs	3.0 Meg
	(4) 1 Meg SIMMs	5.0 Meg

Please note that you can not mix 256K and 1 Meg SIMMs packages on the same GS Sauce card, and that expansion must be performed in (1), (2) or (4) SIMMs modules.

Pricing:

We are offering a limited time "get acquainted" offer to our customers. The GS Sauce card is available from SSSi as:

0K	$89.95 - use your own 256K or 1 Meg SIMMs modules
1 Meg	$179.95
2 Meg	$269.85
4 Meg	$449.75

☞ **We are making a special offer to our Genesys users:**

Buy Genesys and and get a coupon to purchase GS Sauce for:

0K	$79.95 - use your own 256K or 1 Meg SIMMs modules
1 Meg	$159.90
2 Meg	$239.85
4 Meg	$399.75

We hope you will see what an excellant value the GS Sauce card is: low power consumption, SIMMs technology, inexpensive, made in USA and lifetime warranty!

Call or write for seperate 256K and 1 Meg SIMMs modules to upgrade your GS

Order by phone or by mail. Check, money order, MasterCard, Visa and American Express accepted. *Please add $5.00 for S/H*

Simple Software Systems International, Inc.
4612 North Landing Dr.
Marietta, GA 30066
(404) 928-4388

MasterCard VISA AMERICAN EXPRESS

SSSi

PushPin Studios, St. Augustine, FL

by Murphy Sewall

Reprinted from the APPLE PULP (E. Hartford, CT)

Laser, Scanner, FAX, Modem.
National Semiconductor has introduced the NS32GX320 Imaging/Signal processor which can allow a single peripheral to print, scan, send and receive FAXes and function as a modem. Printer-FAX-copiers that include the new chip are expected as early as next fall's Comdex. Ed Pullen, an analyst at San Jose market research firm InfoCorp, predicts an 8 page per minute unit will cost about $2,400. - *PC Week 28 May*

"What If?" Graphics.
Bell Atlantic plans an August release of a Windows 3.0 program called Thinx which allows users to draw or import images, attach numeric values or other attributes to them, and then do "what if" analysis by manipulating the images. Bell Atlantic product manager, Jack Coppley, says Thinx blends drawing tools with database and spreadsheet capabilities. The proposed retail price is $495. - *PC Week 11 June*

Macintosh IIgs?
Maybe John Sculley's reference to a Macintosh IIgs, at AppleVision '90 back in April, wasn't a "Freudian slip." Word from Germany is that Apple dealers are telling their salesmen not to turn away customers asking about the Apple II. Instead the salesman are told to promote the Mac II line which "can be upgraded to the Macintosh IIgs early next year!" - *found in my electronic mailbox*

Low End Macs.
John Sculley is quoted as telling a developers conference recently "We clearly underestimated the market importance for new low-end and laptop Macs. We will catch up by offering both low-end and laptop Macs over the next 12 to 15 months." The long awaited, modular color K-12 Macintosh may be offered as early as October, but the "no compromises" (does that really mean IIgs?) Apple II emulation card may not be ready until next spring (sources say it has existed in one form or another for more than two years, but production cost remains a problem).
- *InfoWorld 4 June, A2-Central June, and my electronic mailbox*

Color PostScript.
Seiko plans to ship a PostScript compatible color thermal printer in August for $7,000. The printer will work on a network and will offer Centronics, RS-232, and Appletalk ports. - *InfoWorld 28 May*

Apple Demos System 7.
Apple engineer Chris Espinosa demonstrated the alpha version of System 7 for the Macintosh at MacAdemia in Rochester, New York at the end of May. The new operating system will, without question require 2 Mbytes of RAM and a hard disk for every machine running it. Apple engineers emphatically deny any plans to make a "cut-down" version for smaller machine (Does that say something about the memory of the K-12 Mac?). Espinosa was quite clear, it will run the Finder and at least one application on a Mac with only 2 Mbytes of RAM. He also is anticipating that Apple may bundle SIMMS with "a good price" (but for less than already is available by mail order). Apple will

make System 7 the operating system bundled with every machine and will run on every machine within a year or so after introduction. - *found in my electronic mailbox*

Micro Channel Extensions.
Sixty-four bit and even 128 bit extensions of IBM's Micro Channel Architecture are under development. When these buses become available, desktop systems will approach the I/O channel capacity of mainframes. - *InfoWorld 11 June*

PM Lite Lives.
Cyco International and GeoWorks continue to work toward Presentation Manager interfaces for DOS even though IBM abandoned the idea last fall. Cyco will begin shipping Autobase, a graphical database system that includes a PM interface in August. GeoDOS from GeoWorks, a multitasking graphic environment that runs in as little as 512K, is scheduled for this Fall (yes, that is the same company that offered a graphic user interface for the Commodore 64 back in antediluvian times - nearly five years ago). - *PC Week 11 June*

PM Programming Difficulties.
Programming in the Presentation Manager environment is said to be so difficult that IBM is hastily porting Motif to OS/2 to keep the Defense Department happy. Motif will permit X Window applications to run under OS/2. - *PC Week 11 June*

Laptop Printer.
Computer Product Plus has a 3.6 pound (including the batteries), 11.5 by 6.75 by 1.125 inch 24 pin thermal printer which prints full width (8.5 inch) paper. The WSP-200 printer is scheduled to ship in August for $349.95. Output quality is said to be comparable with many 24-pin impact printers. Future plans call for the addition of FAX and scanning capabilities. - *InfoWorld 21 May*

Flash (continued).
There's some dispute about how many Macintosh programmers remain working at Beagle Brothers (see last month's column). The original author of Flash has departed, but someone fixed a few bugs and made enough improvements to create version 1.1 (a free upgrade to registered Flash owners). Does building HyperCard stacks count as Mac programming, or must one Think C (4.0)? We'll find out if a substantially enhanced version 2.0 makes it to market "later this year," and if Flash continues to be a "quick, easy, fun, and inexpensive" utility even after System 7.0 is released. - *found in my electronic mailbox*

Automatic, Continuous Backup
Golden Triangle will offer an accelerated SCSI card and Macintosh software that simultaneously writes files to two hard drives as early as this month. The product named DiskTwin is expected to have a "street price" on the order of $500. Robert R. Tillman, a consultant to Golden Triangle, points out that, due to the falling price of hard drives, a user may be able to acquire Disk Twin and two 100 Mbyte drives for about $2,000. - InfoWorld 4 June

---

Another Bert Sighting
Bert Kersey was recently spotted filling up his Porsche at a Speedway station in Detroit not too long ago. Rumor has it that he's taken flight due to the large amount of public attention focused on him recently. He can be reached only by cellular phone, and only when he's in his phone's limited calling area, so he's out of touch with civilization most of the time. - *found in Ross's electronic mailbox*

The Return of PunkWare

Get Control of Yourself, Young Man!

By Jay Jennings

I have some good news and I have some bad news. The good news is that System Disk 5.0 brought with it some fantastic capabilities, including a tool call that installs lots of hip controls in a window with just one line of code! The bad news is that you have to be a contortionist to get the information from those controls during your program.

Way back when I was boy...

In the good old days, if you used a LineEdit control in a dialog box, your program could return the text input by using two toolcalls. If you used a check box, you could get or set the value with only one toolcall. Well, who ever said programming was supposed to be easy all the time? Yeah, we've had it easy up until now, but if you want to use the capabilities of ~NewControl2, get used to the fact that it's a chore to get and set control info now. It's not an impossible task, but it's a pain in the gluteus maximus to try and figure out...which is why I'm writing this: so that your derriere doesn't have to go through the same things that mine did.

This article will introduce the concepts of using the new controls and I'll even include a few subroutines to get you started on the road to Control Mastery.

The source code included in this article is based on some that was written a while back with the help of Eric Mueller, the IIgs editor of 8/16. Eric flew to Kansas City just to help me figure out the difference between a LineEdit Control Record and a LineEdit Edit Record. (Well, that wasn't the only reason, but this code was one of the many things that came out of our marathon hacking sessions.)

Just as a way of introduction, look at listing one. It shows how easy it is to create a bunch of controls using the ~NewControl2 call. You basically pass it the address of a table that is composed of the address of your control templates. It's no big deal. In fact, it makes creating controls a breeze.

Power Programming for Punks

Take a look at listing two for the macros that we'll use for the following routines. I like to put macros like these together as it gives me a sense of working in a high level language, and makes the source code more readable. Notice that the macros I'm using here just parse the parameters and call different subroutines...that way, we can have fairly generic subroutines in our code, yet pull several of them together with a macro for one specific task. If there's one thing I've learned in the past few months, it's to make your code as generic as possible. There are those people who will tell you that by making code generic it won't run as fast. They're right.

But most of the programs I write aren't so speed sensitive that a couple milliseconds are going to make any discernible difference. If you're writing an arcade game, hardcode some stuff. However, if you're writing "normal" applications, you'll find that by making generic routines your coding output will increase in the future. Cut and paste programming will make you a master programmer. And that's what these routines represent: the Lazy Man's Way to Programming.

The first thing we'll tackle was the most complicated for Eric and I to figure out: getting text from an edit line that was created with ~NewControl2. In the following examples, I'm assuming you used ~NewControl2 to create several controls. You can use it to

create just one control at a time, but since most windows have multiple controls, that's what my routines were written for. If you are just creating one control, you'll already have the handle to that extended control record since ~NewControl2 will pass it back to you. When creating more than one control at a time, however, ~NewControl2 doesn't pass back anything useful.

The code in listing two is the subroutine that collects the text from a specified LineEdit control. All the needed parameters are passed via the stack. We need to know three things in order to get the information from the LineEdit control: first, the pointer to the window that the control lives in. (If you want to use the frontmost window, stick a zero in that slot.) Next, we need to know the ID number of the control. Finally, we need the address of a buffer we can stick the text from the control inyo. Make sure you've reserved enough space so that a long piece of text doesn't overwrite whatever happens to fall after the buffer in your source code.

A Closer Look

Okay, let's take a detailed look at what's happening in the GetELine subroutine. The first thing we do is to pull the parameters off the stack and store them in some temporary locations. Note that you must pull the return address off the stack first and then return it else bad things will happen. (Just use the X register to hold the return address temporarily.)

Next we use ~GetCtlHandleFromID to get the handle to the extended control record. This record was created for the control when the ~NewControl2 tool call was used. You can get the extended control record layouts and descriptions from the *Apple IIgs Toolbox Reference, Volume 3*, starting on page 28-87. You're allowed to read these records to get information, but setting the values directly is a **no-no.**

Once we have the handle to the extended control record, we stick it in a direct page location in preparation for dereferencing. After that I-can-now-do-it-with-my-eyes-closed procedure (*editor: speak for yourself, man*), we have a pointer to the LineEdit extended control record. Offset $1C into the extended control record gives us the handle to the LineEdit edit record, and that's what we're looking for. We don't have to dereference that because everything else we need to do can be done with the handle.

We're getting warmer...

The handle we have points to a record that looks quite a lot like that pictured on page 10-5 of the Apple IIgs Toolbox Reference, Volume 1. Ah! We're getting closer to actually finding out what text was entered into the LineEdit control.

We can now use our old friends ~LEGetTextLen to get the length of the entered text, and ~LEGetTextHand to get a handle to the text that was entered. Notice that after we get the length of the entered text, we store it in the first word of the buffer space that we've allocated for the LineEdit text. This is so that when all is said and done, we have a pascal string waiting for us to use. Then we add 1 to the address of the text buffer so that we don't overwrite our length byte with the text from the control.

Finally, once we have the handle to the text, we can use ~HandToPtr to move the text from the LineEdit control to our storage buffer. Ta-da! Several contortions, but once you have a generic routine, you can forget all about how hard it was the first time.

Doing a one eighty

The routine SetELine in listing four allows you to place text into a LineEdit control. It's very similar to the previous routine. It needs one more parameter passed on the stack, and after we get the handle to the LineEdit edit record, we only need to do one call, ~LESetText, to complete the routine.

Listing five shows two routines that can be used to activate or deactivate extended controls in a window. They're fairly simple and only require the window number and the control ID to use. The ~GetCtlHandleFromID call is used as before to get a handle to the specified control. Then ~HiliteControl is used to turn the control on or off. You could combine the two routines into one by pushing one more parameter onto the stack: a zero to activate the control, or 255 to deactivate it. This value would then be used as a parameter for the ~HiliteControl call. For subroutines as small as these, I tend to go with simplification rather than saving a few bytes of code. On larger

projects, you might need the extra room, however.

Listing six contains two routines that you can use to check and uncheck check boxes (say that three times fast!). The routine GetCBValue returns a boolean value on the stack. You could also just load the accumulator with the value and return. At the front of the routine, right before we shove the return address back on the stack, we push a space word.

Now look at the end of the routine. We place the boolean value we're returning in that space that's right above the return address.

A bonus from Uncle Jay

And finally, just to sweeten the pot, there are a couple routines in listing seven that come in handy when you're using GS/OS and Pascal strings in the same program. They're well commented and quite simple, so I won't go into them in detail.

Bugsville

There's a documented bug in ~NewControl2 that you should be aware of (documented in Apple IIgs Tech Note #82). When you use that call, the GrafPort is supposed to be set to the current window but doesn't do that as of System Disk 5.0.2. The way around this is to do a ~SetPort call to the correct window before calling ~NewControl2. That should make everything okey-dokey.

This should be enough information to get you started on the road to control mastery. If this is all the information you've read about extended controls, however, you're doomed to a life of mediocrity. To become a true Apple IIgs programmer requires hard work, self denial, and the purchase of the Apple IIgs Toolbox Reference, Volume Three.

Listing One : NewControl2 Call

```
~NewWindow #WindowTemplate
PullLong WindowPtr
~FrontWindow  ;see what window is in front
_SetPort      ;and set the GrafPort to it
~NewControl2 WindowPtr;#3;#ClientCList
pla
pla           ;just junk the return value
...

ClientCList
     adrl ButtonTemplate
     adrl CheckBoxTemplate
     adrl LineEditTemplate
     adrl 0
```

Listing Two: Get Edit Line Text Subroutine

```
GetEditLineText mac
     PushLong  ]1                  ;window pointer
     PushLong  ]2                      ;control ID
     PushLong  ]3                  ;buffer address
     jsr       GetELine
     eom

SetEditLineText mac
     PushLong  ]1            ;which window to use
     PushLong  ]2            ;edit line control ID
     PushLong  ]3          ;address of text to set
     lda       ]4           ;length of text to set
     and       #$00FF
     pha
     jsr       SetELine
     eom

GetText mac
    ;window ptr, control ID, & storage buffer
     GetEditLine    ]1;]2;]3
     PascalToGS     ]3
     CopyGSString   #GSString;]3;GSString
     eom

PascalToGS     mac
     PushLong  ]1   ;address of the Pascal string
     jsr       Pascal2GS
     eom

GSToPascal     mac
     PushLong  ]1    ;address of the GS/OS string
     jsr       GS2Pascal
     eom

CopyGSString   mac
     PushLong  ]1          ;address of the source
     PushLong  ]2     ;address of the destination
     pea       0  ;zero high word for # of bytes
to move
     lda       ]3        ;number of bytes to move
     inc
     inc ;add two so we move the length word, too
```

```
      pha
      _BlockMove
      eom

GetCheckBoxValue mac
      PushLong  ]1          ;window ptr (or zero)
      PushLong  ]2             ;control ID number
      jsr       GetCBValue
      eom

SetCheckBoxValue mac
      PushLong  ]1          ;window ptr (or zero)
      PushLong  ]2             ;control ID number
      PushWord  ]3          ;1=turn on, 0=turn off
      jsr       SetCBValue
      eom
```

Listing Three : Get Edit Line Text Subroutine

```
*===========================================
*
GetELine ent
*
* this plucks a piece of text from a specified
edit line and stores
* it as a Pascal string in a specified location
*
* enter:
* |                  |
* |- WindowPtr      -|  ptr to wind 0=top window
* |                  |
* |------------------|
* |                  |
* |- IDNumber       -|  ID number of control
* |                  |
* |------------------|
* |                  |
* |- textAddr       -| buff addr for editln txt
* |                  |
*  ------------------
*                     <- stack pointer
*

      plx           ;grab & save the return addr
      PullLong :TextAddr     ;ptr to text buffer
      PullLong :IDNumber      ;control ID number
      PullLong :Window;window we're working with
      phx               ;replace the return addr

      ~GetCtlHandleFromID :Window;:IDNumber
      PullLong dpTemp ;store in direct pg space

      Deref dpTemp;dpTemp2;get ptr from the hndl

      ldy  #$1c;get handle to LE: pg28-100 in
TR#3
      lda  [dpTemp2],y
      sta  :Handle
      iny
      iny
      lda  [dpTemp2],y
      sta  :Handle+2

      ~LEGetTextLen :Handle    ;get length of text

      mov_l :TextAddr;dpTemp  ;move text buff addr
      ldy  #0
      pla      ;now grab length from last toolcall
      sta  [dpTemp],y ;store length in text buffer
      sta  :TextLength  ;and save for use in a sec

      add4 :TextAddr;#1;:TextAddr;pt past len byte

      ~LEGetTextHand :Handle
      PullLong :Handle       ;grab the text handle

      ~HandToPtr :Handle;:TextAddr;:TextLength
        ;copy info to text buffer

      rts

:TextLength     ds    4          ;length of the text
:TextAddr       ds    4         ;address of the text
:Handle         ds    4  ;handle to LineEdit Record
:Window         ds    4  ;window we're working with
:IDNumber       ds    4           ;control ID number
```

Listing Four: Set Edit Line Text Subroutine

```
*===========================================
*
SetELine ent
*
* this places a specified piece of text in a
specified edit line
* enter:
* |                  |
* |- WindowPtr      -| ptr to wnd  0=top window
* |                  |
* |------------------|
* |                  |
* |- IDNumber       -|  ID number of control
* |                  |
* |------------------|
* |                  |
* |- textAddr       -| addr of text for edit line
* |                  |
* |------------------|
*  |  textLength     |  length of text to set
*   ------------------
*                     <- stack pointer
```

```
*
     plx          ;grab & save the return addr
     pla              ;pull off the textLength
     and  #$00FF     ;only allow 256 chars max
     sta  :TextLength  ;how many chars in text
     PullLong :TextAddr   ;pointer to the text
     PullLong :IDNumber     ;control ID number
     PullLong :Window;window we're working with
     phx               ;replace the return addr

     ~GetCtlHandleFromID :Window;:IDNumber
     PullLong dpTemp;store in direct page space

     Deref dpTemp;dpTemp2;get pointer from hndl

     ldy  #$1c;get hndle to LE:pg28-100 in TR#3
     lda  [dpTemp2],y
     sta  :Handle
     iny
     iny
     lda  [dpTemp2],y
     sta  :Handle+2

     ~LESetText :TextAddr;:TextLength;:Handle

     rts

:TextLength    ds   4      ;length of the text
:TextAddr      ds   4     ;address of the text
:Handle        ds   4 ;hndl to LineEdit Record
:Window        ds   4  ;wnd we're working with
:IDNumber      ds   4       ;control ID number
```

Listing Five: Deactivate Control Subroutine

```
*===========================================
*
DeactivateControl ent
*
* enter:     PushLong WindowPtr
*            PushLong ControlID
*
     plx
     PullLong dpTemp2         ;control ID
     PullLong dpTemp          ;window pointer
     phx

     ~GetCtlHandleFromID dpTemp;dpTemp2
     PullLong dpTemp          ;handle to con-
trol

     ~HiliteControl #255;dpTemp

     rts
*===========================================
*
ActivateControl ent
*
* enter:     PushLong WindowPtr
*            PushLong ControlID
*
     plx
     PullLong dpTemp2         ;control ID
     PullLong dpTemp          ;window pointer
     phx

     ~GetCtlHandleFromID dpTemp;dpTemp2
     PullLong dpTemp          ;handle to control

     ~HiliteControl #0;dpTemp

     rts
```

Listing Six: Get CheckBox Value Subroutine

```
*===========================================
*
GetCBValue ent
*
* get the current value of a check box and return
with it on the stack
* enter:
*  |                  |
*  |- WindowPtr      -| ptr to window  0=top wnd
*  |                  |
*  |------------------|
*  |                  |
*  |- IDNumber       -|  ID number of control
*  |                  |
*   ------------------  <- stack
*
* exit:
*  |  value           |boolean value of check box
*   ------------------  <- stack
*
     plx                     ;save return address
     PullLong dpTemp          ;get the control ID
     PullLong dpTemp2     ;get the window pointer
     pha                       ;space for result
     phx                      ;and return address

     ~GetCtlHandleFromID dpTemp2;dpTemp
     PullLong dpTemp ;gethandle to control record

     Deref dpTemp;dpTemp2

     ldy #$12     ;offset into the control record
     lda [dpTemp2],y          ;get the item value
     sta 3,s           ;and save it for later use
     rts
```

```
*===========================================
*
SetCBValue ent
*
* set a check box to a specified value. 0 = not
checked  1 = checked
* enter:
* |                  |
* |- WindowPtr      -| ptr to window 0=top wnd
* |                  |
* |------------------|
* |                  |
* |- IDNumber       -|  ID number of control
* |                  |
* |------------------|
* |  newCtlValue     | val for ck box (0 or 1)
*  ------------------  <- stack
*
* exit:
*
     plx                  ;save return address
     ply          ;grab new value for checkbox
     PullLong dpTemp       ;get the control ID
     PullLong dpTemp2      ;get the window ptr
     phx                       ;return address
     phy                     ;new value for CB

     ~GetCtlHandleFromID dpTemp2;dpTemp
     PullLong dpTemp   ;hndl to control record

     Deref dpTemp;dpTemp2

     ldy #$12      ;offset into control record
     pla    ;retrieve the new value from stack
     sta [dpTemp2],y       ;set the item value
     rts
```

Listing Seven : GSOS 2 Pascal String Covert

```
*===========================================
*
GS2Pascal ent
*
* turn a GS/OS string into a Pascal string.
*
* enter:
*   |                  |
*   |- StringAddr     -|    addr to GS/OS string
*   |                  |
*    ------------------  <- stack pointer

     plx                    ;grab return addr
     PullLong dpTemp ;get addr to GS/OS string
     phx                   ;replace return addr

     lda [dpTemp]             ;get the length word
     and #$00FF      ;strip off anything over 255
     sta PascalString       ;save the length byte
     tax
     beq :Zero             ;don't do null strings

     ldy #2           ;start past the length word
     sep #$20
]loop
     lda [dpTemp],y             ;grab a character
     sta PascalString-1,y       ;and save it anew
     iny
     dex
     bne ]loop
     rep #$20
:Zero
     rts

PascalString ent
     ds 255  ;255 = max length of a Pascal string

*=======================================
*
Pascal2GS ent
*
* turn a Pascal string into a GS/OS string and
store result at GSString
*
* enter:
* |                  |
* |- StringAddr     -|    address to pascal string
* |                  |
*  ------------------  <- stack pointer

     plx                        ;grab return addr
     PullLong dpTemp      ;get addr to Pascal str
     phx                     ;replace return addr

     lda [dpTemp]             ;get the length byte
     and #$00FF   ;strip garbage from 16bit accum
     sta GSString  ; and save the new length word
     tax
     beq :Zero             ;don't do null strings
     ldy #1               ;start past length byte
     shorta
]loop
     lda [dpTemp],y              ;get a character
     sta GSString+1,y    ;save it in the new spot
     iny
     dex
     bne ]loop
     longa
:Zero
     rts

GSString ent
     ds 255        ;max allowed for Pascal string
```

From the House of Ariel

• *8/16 on Disk* •

The magazine you are now holding in your hands is but a subset of the material on the *8/16* disk. We have combed the BBS's and data services across the country to collect the best of the public domain and shareware offerings for programmers. Not only that, but we have extra articles and source code written by our staff. With DLT16 and DLT8 (**D**isplay **L**auncher **T**hingamajigs) to guide you, you can read articles, display graphics, and even launch applications. **NOTE: DLT16 requires GS/OS v 5.02 on your system.**

Highlights (so far every disk has had more than 650K of material!):

- **March '90:** 8 bit - the entire source code to Floyd Zink's Binary Library Utility. 16 bit - Bill Tudor's fantastic InitMaster CDEV, Parik Rao's Orca/APW utilities
- **April '90:** 8 bit - SoftWorks, an AppleWorks™ filecard interface for Applesoft programs, the source code to Bruce Mah's File Attribute Zapper. 16 bit - More Orca and APW utilities, Phil Doto's APF viewer
- **May '90:** 8 bit - Tom Hoover's AppleWorks Style Line Input. 16 bit - Bryan Pietrzak's shell utilities for Orca/APW, Steve Lepisto's "Illusions of Motion".
- **June '90:** 8 bit - 3D graphics package, MicroDot™ Demo, DiskWorks, 80 column screen editor. 16 bit - Assembly Source Code Converter (shareware), Install DA (on the fly; by our our own Eric Mueller), Find File source code.

1 year - $69.95 6 months - $39.95 3 months - $21 Individual disks are $8.00 each

• *Shem The Penman's Guide To Interactive Fiction* •

This is undoubtedly my personal favorite of all our software offerings. First of all, it is FUN. Second of all it is a very well organized, well written, and well programmed introduction to programming interactive fiction. It is, in fact, the only package of its kind I've ever seen!

Author Chet Day is a professional writer (go buy *Hacker* at your nearest book store!) and an educator who is as conerned with the *content* of your interactive fiction program as with the form. This package is fun, entertaining, and useful. It includes Applesoft, ZBasic, and Micol Advanced Basic "shells" which will drive your creations - **$39.95 (both 5.25" and 3.5" disks supplied).** P.S. The advantage to the ZBasic and Micol versions is that with the easy integration of text and graphics provided in those langauges, you can easily load a graphic and overlay text in the appropriate spots.

• *ProTools*™ •

Fast approaching its first birthday, our ProTools library for ZBasic programmers has grown into a mature and powerful product. It's bigger than ever, too. *inCider's* Joe Abernathy called it, "...the only way to go for ZBasic programmers."

ProTools includes a text based *and* a double high resolution graphics based desktop interface (pull-down menus, windows, mouse tracking, etc.) Both desktops support quick-key equivalents for menu items, too! We've added a *third* desktop package in version 2.5 of ProTools, too. This one is mouseless, meaning that it is entirely keyboard driven and therefore much more compact than its predecessors.

ProTools contains literally *scores* of additional functions and routines, including:

- FRAME.FN
- GETMACHID
- SAVE_SCREEN
- DATETIME
- ONLINE
- SETSPEED
- SMART.INPUT.FN
- GETKEY.FN
- DIALOG
- BAR CHART
- PASSWORD
- VERTMENU
- SCROLL.MENU.FN
- SCREENDUMP80
- CRYPT
- LINE GRAPH
- READTEXT
- PATHCK

ProTools is $39.95 (your choice of 3.5" or 5.25" disks).

• *Back issues of The Sourceror's Apprentice* •

Ross's Recommendations:

8 bit:

Feb '89 - Relocation Without Dislocation, by Karl Bunker
...techniques for writing relocatable 8 bit code
Jan, Mar, Apr, Aug '89 - The Applesoft Connection Parts 1-4, by Jerry Kindall
...using the ampersand vector and internal Applesoft routines. A classic series.
Jun '89 - Peeking at Auxiliary Memory: A Monitor Utility, by Matthew Neuberg
...lets the monitor display aux mem, an invaluable 128K programming tool.
Sep '89 - Getting More Value(s) From Your Game Port, Eric Soldan
...increase range of values returned by a joystick for DHR coordinates, etc.

16 bit:

Jan '89 - Programming with Class 1, by Jay Jennings
...an introduction to GS/OS class 1 calls
Mar & Jun '89 - Vectored Joystick Programming, by Stephen Lepisto
...a technique for increasing responsiveness in reading the joystick
July '89 - Making a List (and checking it twice), by Ross W. Lambert
...an introduction to the GS List Manager
Sep '89 - Generic Start II, The Sequel, by Jay Jennings
...an introduction to the new start up song and dance for new system software
Jan '90 - Trapping Tricky Tool Errors, by Jay Jennings
...a classy programmer's error trap for the GS.

All back issues are $3.00 each (postage and handling included except for non-North American orders. Those of you on other shores please add $1.50 extra per issue).

Our guarantee: Ariel Publishing guarantees your satisfaction with our entire product line (software *and* publications). If you are *ever* dissatisfied with one of our products, we will cheerfully refund the amount you paid on your request. To order, just write to: **Ariel Publishing, Box 398, Pateros, WA 98846 or call (509) 923-2249.**

Hired Guns

8/16 is providing a free service to all programmers (who are subscribers!): placement of a complimentary "situation wanted" ad. If you're available for hire and looking for a programming job (from full-time to freelance), a listing in this directory is your ticket to work. The ads are open to both 8 and 16 bit authors and are limited to 120 words or less. Be sure to give your address, phone number, and email addresses, and specify how much of a job you're after (part-time? full-time? royalty-based? etc). Send it to Situation Wanted, c/o Ariel Publishing, Box 398, Pateros, WA 98846

David Ely. 4567 W. 159th St. Lawndale, CA 90260. 213-371-4350 eves. or leave message. GEnie: [DDELY], AOL: "DaveEly". Experienced in 8 and 16 bit assembly, C, Forth and BASIC. Available for hourly or flat fee contract work on all Apple II platforms (IIgs preferred). Have experience in writing desktop and classical applications in 8 or 16 bit environments, hardware and firmware interfacing, patching and program maintenance. Will work individually or as a part if a group.

Jeff Holcomb, 18250 Marsh Ln, #515, Dallas, Tx 75287. (214) 306-0710, leave message. GEnie: [Applied.Eng], AOL: "AE Jeff". I am looking for part-time work in my spare time. I prefer 16-bit programs but I am familiar with 8-bit. Strengths are GS/OS, desktop applications, and sound programming. I have also worked with hardware/firmware, desk accessories, CDevs, and inits.

Tom Hoover, Rt 1 Box 362, Lorena, TX, 76655, 817-752-9731 (day), 817-666-7605 (night). GEnie: Tom-Hoover; AOL: THoover; Pro-Beagle, Pro-APA, or Pro-Carolina: thoover. Interests/strengths are 8-bit utility programs, including TimeOut(tm) applications, written in assembly language. Looking for "part-time" work only, to be done in my spare time.

Jay Jennings, 14-9125 Robinson #2A, Overland Park, KS, 66212. (913) 642-5396 late evenings or early mornings. GEnie: [A2.JAY] or [PUNKWARE]. Apple IIgs assembly language programmer. Looking for short term projects, typically 2-4 weeks. Could be convinced to do longer projects in some cases. Familiar with console, modem, and network programming, desk accessories, programming utilities, data bases, etc. GS/OS only. No DOS 3.3 and no 8-bit (unless the money is extremely good and there's a company car involved).

Jim Lazar, 1109 Niesen Road, Port Washington, WI 53074, 414-284-4838 nights, 414-781-6700 days. AOL: "WinkieJim", GEnie: [WINKIEJIM]. Strengths include: GS/OS and ProDOS 8 work, desktop applications, CDAs, NDAs, INITs. Prefer working in 6502 or 65816 Assembly. Have experience with large and small programs, utilities, games, disk copy routines and writing documentation. Nibble, inCider and Call-A.P.P.L.E. have published my work. Prefer 16-bit, but will do 8-bit work. Type of work depends on the situation, would consider full-time for career move/benefits, otherwise 25 hrs/month (flexible).

Stephen P. Lepisto, 12907 Strathern St., N. Hollywood, CA 91605, 818-503-2939. GEnie: S.LEPISTO. Available for full-time and part-time contract work (flat rate or royalties). Experienced in 6502 to 65816 assembly, BASIC and C. Can work in these or quickly learn new languages and hardware (some experience with UNIX, MS-DOS, 8086 assembly). Experience in games, utilities, educational, applications. Lots of experience in porting programs to Apples. Programmed Hacker II (64k Apple II), Labyrinth (128k Apple), Firepower GS and others. Can also write technical articles.

Chris McKinsey, 3401 Alder Drive, Tacoma, WA, 98439, 206-588-7985, GEnie: C.MCKINSEY. Experience in programming 16-bit (65c816) games. Strengths include complex super hi-res animation, sound work (digitized and sequenced), and firmware. Looking for new IIgs game to develop or the porting of games from other computers to the IIgs.

Eric Mueller, 2760 Roundtop Drive, Colorado Springs, CO, 80918, 719-548-8295 anytime. GEnie: [A2PRO.ERIC], CIS: 73567,1656, AO: "A2Pro Eric". Strengths include GS/OS and ProDOS 8 work, console, and modem I/O, working with hardware/firmware, desktop applications, desk accessories. Can also do tool patches, INITs, whatever. Don't call me for complex animation or sound work. Have experience working with others on programs, and on large applications. References available. Prefer 16 bit stuff always. Looking for _very_ small (less than 25 hrs/month) jobs right now.

Bryan Pietrzak, 4313 West 207th St, Matteson, Il, 60443, (708) 748-6363, or (217) 356-4351. GEnie: B.PIETRZAK1. Strengths include database design and data structures (hashing, etc) and

Continued on p. 43

Ariel Publishing
P.O.Box 398
Pateros, WA 98846
(509) 923-2249

Please clip and mail this form to us. rather fax it, have your machine cal (509) 689-3136.

The First Annual 8/16 Subscriber Survey

For each question, circle the response that is nearest to your own point of view.

1.) Overall, I rate 8/16's value as... Excellent Good Fair Poor

2.) The subject matter of the articles has been... Excellent Good Fair Poor

3.) The quality of the writing and instruction in the articles has been... Excellent Good Fair Poor

4.) The general editorial tone of 8/16 is:

too silly and flippant | a little too light | about right | too serious

5.) I find the level of difficulty of the magazine:

way too hard to understand | about right, clear but challenging | not nearly meaty enough

6.) I find the layout of 8/16:

too spread out - scrunch it all up so we get more | too scrunched up and unartistic

about right, a decent tradeoff between quantity and readability

7.) The balance between GS and 8 bit programming is...

too slanted in favor of the 8 bit Apples | about right | too slanted in favor of the GS

8.) I'd like to see an article about:

The First Annual Subscriber Survey (continued)

9.) My age is: under 20 20-29 30-39 40-49

10.) My annual income is approximately: under $20,000 $20,000 - 29,999 $30,000-39,999 $40,000 - $49,000

11.) In the next year I expect to purchase software (for either business or personal use) to the tune of:

less than $50 $50-$149 $149-$249 $250-$399 $400+

12.) The software I purchase will most like fall into the following categories (Prioritize if more than one applies, that is, make the most likely #1, the next most likely, #2, etc.):

_____software development tools (languages, prototopyers, etc.)
_____graphics
_____educational
_____database
_____spreadsheet
_____word processor

13.) I expect to purchase the following hardware products in the next year (for either business or personal use - circle all that apply):

an Apple IIGS
an Apple IIe or IIc+
a Laser (Apple II clone)
a Macintosh
an accelerator board
a sound digitizer
a hard drive
a modem
a dot matrix printer
a laser printer
a video scanner
a MIDI board
RAM chips
a floppy drive

14.) I currently own the following computers (circle all that apply):

Apple IIe, IIc, IIc+
Apple IIgs
Macintosh Plus
Macintosh SE
Macintosh SE/30, II, IIci, IIcx, or IIfx
an XT class IBM or compatible
an AT class IBM or compatible
other IBM compatible

15.) My favorite 8/16 article to date has been:

More Hired Guns...

Lane Roath, Ideas From the Deep, 309 Oak Ridge Lane, Haughton, LA 71037. (318) 949-8264 (leave message with phone number!) or (318) 221-5134 (work). GEnie: L.Roath, Delphi: LRoath. Available for part time work, large or small for any of the Apple II line, especially the IIgs. Specializing in disk I/O graphics and application programming. Wrote Dark Castle GS, Disk Utility Package, WordWorks WP, Project Manager, DeepDOS, LaneDOS, etc. including documentation. Currently work for Softdisk G-S. Work only in Assembler.

Steve Stephenson (Synesis Systems), 2628 E. Isabella, Mesa, AZ, 85204, 602-926-8284, anytime. GEnie: [S-STEPHENSON], AOL: "Steve S816". Available for projects large or small on contract and/or royalty basis. Experienced in programming all Apple II computers (prefer IIGS), documentation writing/editing and project management. Have expertise in utilities, desk accessories, drivers, diagnostics, patching, modifying, and hardware level interfacing. Willing to maintain or customize your existing program. Work only in assembly language. Authored SQUIRT and Checkmate Technology's AppleWorks Expander, managed the ProTERM(tm) project, and co-invented MemorySaver(tm) [patent pending].

Jonah Stich, 6 Lafayette West, Princeton, NJ, 08540. (609) 683-1396, after 3:30 or on weekends. America OnLine (preferred): JonahS; GEnie: J.STICH1; InterNET: jonah@amos.ucsd.edu. Have been programming Apples for 7 years, and can speak Assembly (primary language), C, and Pascal. Currently working on the GS, extremely skilled in graphics, animation, and sound, as well as all aspects of toolbox programming. Prefer to work alone or with one or two others. Can spend about 125 hours a month on projects.

Loren W. Wright, 6 Addison Road, Nashua, NH 03062, (603)-891-2331. GEnie: [L.WRIGHT2]. Lots of experience in 6502 assembly, BASIC, C, Pascal, and PLM on a wide variety of machines: Apple II, IIgs, C64, VIC20, PET, Wang OIS. Some IIgs desktop programming. Have done several C64<>Apple program conversions. Numerous articles and regular columns in Nibble and MICRO magazines. Product reviews and beta testing. Specialties include user interface, graphics, and printer graphics. Looking for full-time work in New England and/or at-home contract work.

Advertiser Index

The Sensational Lasers

Apple IIe/IIc Compatible

SALE **$345** *Includes 10 free software programs!*

New! **Now Includes COPY II PLUS®**

The Laser 128® features full Apple® II compatibility with an internal disk drive, serial, parallel, modem, and mouse ports. When you're ready to expand your system, there's an external drive port and expansion slot. The Laser 128 even includes 10 free software programs! Take advantage of this exceptional value today.....***$345***

Super High Speed Option!

only **$385**

The LASER 128EX has all the features of the LASER 128, plus a triple speed processor and memory expansion to 1MB $385.00

The LASER 128EX/2 has all the features of the LASER 128EX, plus MIDI, Clock and Daisy Chain Drive Controller $420.00

DISK DRIVES

* 5.25 LASER/Apple 11c ..... $ 99.00
* 5.25 LASER/Apple 11e ..... $ 99.00
* 3.50 LASER/Apple 800K $179.00
* 5.25 LASER Daisy Chain ... New! $109.00
* 3.50 LASER Daisy Chain ... New! $179.00

Save Money by Buying a Complete Package!

THE STAR a LASER 128 Computer with 12" Monochrome Monitor and the LASER 145E Printer $620.00

THE SUPERSTAR a LASER 128 Computer with 14" RGB Color Monitor and the LASER 145E Printer $785.00

ACCESSORIES

* 12" Monochrome Monitor ..... $ 89.00
* 14" RGB Color Monitor $249.00
* LASER 190E Printer $219.00
* LASER 145E Printer ..... New! $189.00
* Mouse ..... $ 59.00
* Joystick (3) Button ..... $ 29.00
* 1200/2400 Baud Modem Auto $129.00

USA MICRO

YOUR DIRECT SOURCE FOR APPLE AND IBM COMPATIBLE COMPUTERS

MasterCard VISA

2888 Bluff Street, Suite 257 • Boulder, CO. 80301
Add 3% Shipping • Colorado Residents Add 3% Tax

Your satisfaction is our guarantee!

Phone Orders: 1-800-654-5426
8 - 5 Mountain Time • No Surcharge on Visa or MasterCard Orders!
Customer Service 1-800-537-8596 • In Colorado (303) 938-9089
FAX Orders: 1-303-939-9839