October, 1990
Volume 1,
No. 8

8/16

The Journal of Apple II Programming $4.00

Safe X (CMDs) for HyperStudio™

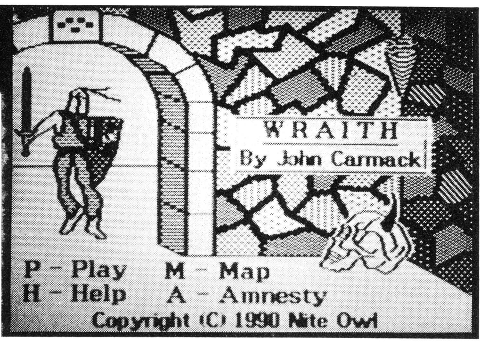
Ariel Publishing P.O.Box 398
Pateros, WA 988
(509) 923-2249

New

WRAITH

Adventure

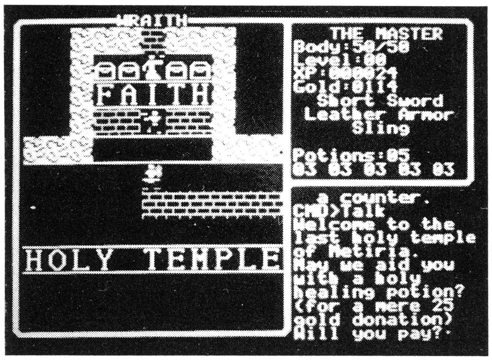
Special Introductory Price $9.95



The **Devil's Demise** - is an exceptional graphic adventure game that 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 Araithia. 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.

An excellent adventure for Apple IIe, IIc, and IIGS computers with a 3.5" drive. It has a retail price of $14.95, but you can take advantage of our introductory offer and order it direct from Nite Owl for only $9.95 before 12/31/90.



```
THE MASTER
Body:50/50
Level:00
XP:000024
Gold:0114
 Short Sword
 Leather Armor
 Sling

Potions:05
03 03 03 03 03
```

```
a counter.
CMD>Talk
Welcome to the
last holy temple
of Aetiria.
May we aid you
with a holy
healing potion?
(For a mere 25
gold donation)
Will you pay?
```

***"I have never in my life seen a better way to spend such a modest amount of money"*,**

writes Neil Shapiro in his review of WRAITH in the July 1990 issue of Nibble magazine.

√√√ *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, an Appleworks data file that lists the available fonts, and picture files that let you view the various fonts.

This collection includes over 8 Mb of fonts. Due to the large volume of this collection, a hard disk is highly recommended. Only $39 for this valuable collection.

In Depth:

Close Out!

It was more than just "Bad News" when Tech Alliance ceased publication of Call -A.P.P.L.E. magazine. It was a major loss of technical information and support for the Apple II. In order to help keep some of this information available, we have acquired the last remaining copies of their manual, "**All About Applesoft** - In Depth".

It is written for the highly technical, Applesoft and Assembly language programmer. It includes a list of internal entry points in the Applesoft ROM and describes how to use them. This classic is now out of print, in short supply, and available from Nite Owl for $20. Limit 1

Keep it Cool and Quiet

When you start adding more memory and additional interface cards, your IIGS computer can overheat. This can cause malfunctions and shorten the life of your computer.

The GS Super Cooler fan fastens to the internal power supply and is powered from the standard fan jack on the motherboard. They are easily installed, cause no audio line interference, and they are quieter and less expensive than other alternatives. They are available for $24 each.

Call: (913) 362-9898 FAX: (913) 362-5798

Satisfaction Guaranteed: If you are not completely satisfied with anything you order from Nite Owl, return it within 30 days for a prompt refund or replacement.

Use this handy Cut & Paste address label for fast service

Nite Owl Productions
5734 Lamar Avenue A
Mission, KS 66202-2646

• School Purchase Orders are Welcome •

Cash, Check, Money Order | VISA | Master Card | Purchase Order

Ship to:

____

____

____

____

Telephone #: ______________________

Credit Card or PO#	Expires

Quantity	Description	Price	Amount
	Slide-On Battery Kits	$ 14.95	
	Nite Owl Journal 3.5	$ 9.95	
	WRAITH Adventure	$ 9.95	
	GS Super Cooler Fan	$ 24.00	
	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	

Prices may Change Without Notice

8/16

Copyright (C) 1990, Ariel Publishing, Most Rights Reserved

Publisher & Editor-in-Chief	Ross W. Lambert
Apple IIgs Editor	Eric Mueller
Classic Apple Editor	Jerry Kindall
Contributing Editors	Jay Jennings
	David Gauger
	Steve Stephenson
	Mike Westerfield
	Cecil Fretwell
Subscription Services	Tamara Lambert
	Karen Redfield

Subscription prices in US dollars:

• magazine	1 year $32	2 years $60
• monthly disk	1 year $69.95	2 years $129

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) or (509) 689-3136 (fax).

The Publisher's Pen

by Ross W. Lambert

More Conferencing, C Cecil Code, & Marketing 101

I was particularly impressed with Dennis Doms' summary, report, and analysis of the A2-Central Developer's Conference. If you want to read an absolutely incisive review of it all, call A2-Central (913/469-6502) and ask for a copy of their September issue.

I was insanely jealous and in awe of Dennis's ability to recall detail (and stay awake) until I remembered that he could probably review the video tapes of everything.

More C From Cecil

When Cecil Fretwell gets into something, he *really* gets into something. Not only did he translate "Hello World" from the Lichty and Eyes book (*Programming the Apple IIGS in Assembly Language*) into C, but he has now also translated Sandy Mossberg's "Skeleton Desktop Application" from our late, great forebear, *CALL A.P.P.L.E.* Our spiritual parent magazine ran parts one through three of Sandy's five part series. To my knowledge, Cecil's disk is the only place to find parts four and five. Plus it serves as a wonderful reference for anyone learning C. This new disk is available directly from Cecil for a mere $20. Write Uncle Cecil at: 2605 Highview Avenue, Waterloo, Iowa, 50702.

Marketing for Small Developers Who Want to Survive to be Big Developers

I've seen 'em come and I've seen 'em go. And you wanna know what? They mostly go.

With that little homily as background, let's begin my promised foray into marketing.

You may wonder, at first, why I'd have any inclination to share my "secrets". Well, first of all, there really aren't any secrets. As Solomon said about 2500 years ago, "There is nothing new under the sun."

Combine that with the fact that most folks think I'm off my rocker, anyway, and I don't have much to worry about.

And if that is not enough, it is important to the Apple II market that developers of *really good* products also know how to make money at it. The market itself will pass judgement when you combine excellent marketing with a crummy product. I just want to prevent good products from disappearing due to poor marketing.

"...it is important to the Apple II market that developers of* really good *products also know how to make money at it."

Advice You'll Probably Ignore

I'd like to begin here in the same manner I did in Kansas: by offering some general purpose, seemingly unrelated bits of advice. This may seem pretty bizarre, but I honestly think it is the most valuable portion of my marketing presentations. Of course, the offerer of unsolicited advice usually does overestimate its worth!

Here are the bits, in no particular order...

➡ Read the book of John in the New Testament. Religious considerations aside, it is an excellent treatise on starting a worldwide movement on a tiny budget. Although my born-again brethren will probably call me crass and materialistic, I nevertheless think there are some important lessons for marketers in there. Do you think it is some kind of semantic accident that Apple employed "evangelists" to spread the Macintosh gospel?

➡ Read *Sun Tzu and the Art of War*. Sun Tzu was the ancient Chinese version of Lee Iococca - except that the stakes were higher for Mr. Sun.

➡ No debt for expansion for at least two years. It'll take you that long to figure out how to spend money without wasting it. Make your mistakes cheap ones, relatively speaking.

➡ It is better to be underestimated than overestimated. People are more likely to help you if A) they like you, and B) they don't perceive you as a threat. Acting the big shot can impress the mush brains in the short run. It'll make you broke in the long run.

➡ Hold your tongue, the wheel turns. That's not to say that you shouldn't stand firm on your principles, but that can be done without spouting off. Bill Gates can spout off and get away with it. As far as I can ascertain, he's the only software developer around in a position to do so.

Why Publish Yourself?

A) No one cares as much about your product as you do. You can rest assured that it will not languish unpromoted. This is a real problem at times because a publisher's marketing is often not what the author thinks it should be.

B) Change is a constant in this industry and appearances can be deceiving. The folks flashing money right and left today are likely to be filing Chapter 13's tomorrow.

Legal hassles abound, too. Like what happens when your biggest competitor buys out your publisher?

Furthermore, some publishers I've seen "sign" as many products as possible in the hope that one will turn out to be a big winner. It makes them look good to have such a thick catalog. It makes *you* look bad, however, when the programs sold with yours are trash.

C) You make more money per unit sold. It is the rare developer who can command 20% of the purchase price. Most first-time authors are lucky to get something in the 5-10% range. If you sell your own creations, the amount you get depends on you and nobody else.

This is not to say that there are not serious drawbacks to self-publishing. It is almost guaranteed that you'll not move as many products since you don't have the ad budget of the big boys. And you'll need to learn a few things and spend about 10-20% of your time working on marketing related projects.

Serious Bun Covering Time

By the way, I'm **not** suggesting that there are not publishers around who are trustworthy. If I had time to develop a consumer package I'd be quite comfortable having Roger Wagner Publishing or Jem Software (Randy Brandt's new Co.) produce it. If I had a language or heavy duty programming product I'd consider The Byte Works in a heartbeat. And there are many more. I've really enjoyed our working relationship with Night Owl Productions and Bob Shofstall, too. There are many others, too, I'm sure. I just don't know the folks personally.

Why the Apple II Market?

In my not-so-humble opinion, the Apple II market is perfect for small developers.

The main reason is that the market is mature - Apple II users know what they want and can smell a trashy product a mile away. For this reason, small developers can compete with the big boys' ad bucks quite effectively.

In the Mac market, on the other hand, if you can't splash a four color ad across two pages of *MacUser*, you're nowhere. Never mind that such a venture would run you $10,000+ per month.

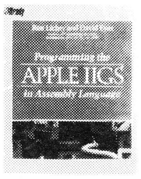
I don't know if you're a gambler, but the Apple II market also has the potential for a fairly decent turnaround. I can't predict the future very well yet, but if Apple puts *any* marketing muscle behind the IIgs, we who have developed really good products for the II will be in position to benefit right away.

There's a really big assumption underlying this entire series: our product and our customer support must be very, very good. Good marketing just allows us to reach our potential. As we improve our product and product support, our potential improves.

Here's hoping we all can do just that in the coming months.

== Ross ==

Program the IIGS!



Programming the Apple IIGS in Assembly Language by Ron Lichty and David Eyes. The easiest-to-follow step-by-step guide to creating full-fledged Apple IIGS applications. Develop **Hello, World** from an 8-line program that prints on the text screen to a full-blown desktop program with menu bar, dialogs, icons, and multiple, sizeable, scrollable windows! Thorough reference section. 550 pages. "Addictive... the more I read, the more fascinated I became... In my opinion, this book will fill a big gap in the world of the Apple IIGS." (*Call-APPLE* technical editor Cecil Fretwell) "A must for would-be Apple IIGS programmers... a jump start for beginners and experienced programmers alike." (*Nibble* editor David Krathwohl) "This book belongs in every Apple IIGS programmer's library." (Diversi-software author/publisher Bill Basham) $32 postpaid

Hello, World disks (code from the book, on disk):
APW/ORCAM $20; Merlin $10; C (APW/ORCA) $20

ORCA/M Assembler (Byte Works) $46 postpaid
ORCA C Compiler (Byte Works) $84 postpaid

Calif: add 7% tax. No VISA/MC. Send SASE for details.
Foreign, add: Canada $2; Europe $14 (air); Asia $20 (air)

Ron Lichty (8), POB 27262, San Francisco, CA 94127

AppleSoft to ZBasic Conversion Program

A to Z: AppleSoft to ZBasic is a Compiled ZBasic program and will run on any 64k Apple II. It reads an AppleSoft program file and creates an ASCII text file that can be read by ZBasic, making syntax changes where possible.

Ex: **VTAB 10:HTAB 25** changed to **LOCATE 24,9**

Line numbers are optional. Labels are created by putting double quote's around the old line number, that any GOTO or GOSUB points to.

• Covert's AppleSoft programs to ZBasic (including DOS statements).
• Conversion Approaches 98%.
• Dramatically reduces conversion time.
• Add's DIM statements for all variables.
• Line number or Labels are optional.

Please send ____ Copies at $29.95 ea.
Add $3.00 for Postage and Handling - Outside the USA add $10.00
Ohio residents add 5.5% sales tax. There is no delay on check orders.

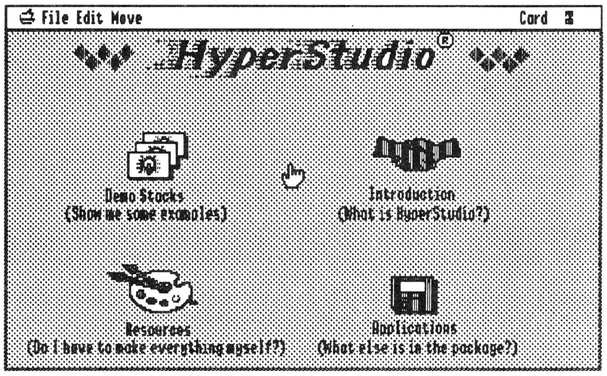
Enclosed $________ By Check __ Money Order __
Name ____________________
Address ____________________
City ____________ State _____ Zip ________

Bringardner Data Products, 1736 E. North Broadway
Columbus, Ohio 43224-4364

Everything You Ever Wanted To Know About X (CMDS)*

*but were afraid to ask

by Eric Mueller



Editor: There's no doubt that HyperStudio™ from Roger Wagner Publishing is one of the "defining" products for the IIgs. I've even heard Apple engineers sing its praises. Still, I have been reluctant to dive into the XCMD game in part because of new terminology. Catch-phrases like "HyperStudio Information Block" turned my palms clammy. I was quite pleased, therefore, to receive Eric's clear and concise treatise on the program's innards. In a nutshell: it's easier than I thought. == Ross ==

HyperStudio XCMDs (pronounced "x-commands") give you the power to do anything at all within the HyperStudio environment.

That's quite a sweeping statement--but completely true. XCMDs allow you to define new actions for buttons, the heart of HyperStudio. By simply choosing "Trigger an XCMD..." in the button actions dialog, when creating any button, you can cause a mouse-click to give control to an external module on the disk.

Assumptions...

This article assumes you're somewhat familiar with HyperStudio, and how buttons and cards are used within that environment. You should also be somewhat familiar with assembly language. The source code provided in Merlin 16+ format but it should be a simple matter to convert it to APW or ORCA/M.

Safe Hex

HyperStudio XCMD files must be named "HS.XCMD", have their filetype set to $BC (Generic Load File) and be located in the same directory as the stack you wish to use them with. Currently, only one XCMD per stack is allowed, unless you're using Ken Kashmarek's Master XCMD package. This package, which gives you a master "HS.XCMD" file and several other XCMDs as modules (named "XCMD.BEEP", "XCMD.VIDEO", etc), allows you to use more than one XCMD per stack. I highly recommend it for XCMD developers; it makes development very simple by allowing you to test several XCMDs within the same stack.

An XCMD can do almost anything it wishes--from beeping the speaker to accessing a videodisc player to moving disk files around to presenting entirely new screen displays independent of HyperStudio. As a matter of fact, two of those XCMDs already exist (the one to beep the speaker and the one to access a videodisc player). Other XCMDs already written include programs that display dialog boxes, control an Apple Video Overlay Card, change the border color, play audio tracks on a CD-ROM drive, and move to a random card number. You can get more XCMDs from the commercial information services (such as GEnie), or possibly your local user group may have a few in their software library.

In this article, I'll be presenting four new XCMDs for you to experiment with: one to beep the speaker a single time (called Beep), an XCMD to beep the speaker as many times as you like, depending on the value you pass it (called Beep2), an XCMD to dial a Hayes-compatible modem attached to your modem port (Dial), and an XCMD to let HyperStudio function as an automatic appointment book (Date). These XCMDs took me various amounts of time to develop, from 15 minutes for the BEEP command to three hours for both the DATE and DIAL XCMDs.

The Environment

Developing these was an absolute pleasure--between the speed of Merlin and being able to hop in an out of HyperStudio, I realized what a nice testing environment HyperStudio provided to me for playing with chunks of code. I could work with TextEdit fields and other control types, sound, the super hi-res screen, the HyperStudio stack environment (where there are several cards and I can move to any of them), and much more.

If I find myself writing code that returns a correct or incorrect value (for example), it's much easier for me to test it as an XCMD within HyperStudio than it is for me to write a tool startup procedure and event loop and build all the associated structures (menus, windows, etc) just to have a nice shell to play around it. And that's how I think of HyperStudio: as something of a feature-rich 'super shell'. It's very easy to hook up and trade data with it, and that makes testing code a very simple matter--much easier than tearing up another program to transplant your test material.

The HS.DEMO disk (in the HyperStudio package) has a directory called "Xcmds" with the files "XCMD.DOC" and several directories. The "XCMD.DOC" file, written by myself in April of last year and updated that September by Ken Kashmarek, briefly outlines the XCMD specifications. The directories contain several demonstration XCMDs (with Merlin 16+ source code) written in assembly language: Find, Port2, Video, and Exerciser. Find, Port2, and Video each provide various extensions to HyperStudio, while Exerciser shows what HyperStudio passes to an XCMD and what the XCMD can pass back and even control in HyperStudio.

Exerciser is published in the HyperStudio package as Merlin 16+ source code, but is also available in ORCA/C, ORCA/Pascal, TML Pascal, and ORCA/M assembly versions. These other versions can be found on a local information service or user group library disk near you, or you can order them from Roger Wagner Publishing for $10. Additionally, they will appear on this month's *8/16 On Disk*, so if you're a disk subscriber, you'll have them.

Quick on the Trigger

When a user selects "Trigger an XCMD" in the HyperStudio button action, they are prompted to type in a line of text to pass to the XCMD module. This string is inserted into the button's definition, within the stack, so that different buttons may pass different strings to the XCMD. If you're using the Master XCMD system, this string should start with the name of the XCMD you wish to use (for example, "BEEP 4" or "DATE +E" instead of just "4" or "+E", when not using the Master XCMD).

The XCMD will be passed control when the button is clicked. Upon entry, the accumulator will contain the user ID of the XCMD module, and the Y and X registers will contain the low and high words of a long pointer to the HyperStudio information block (c.f. Table 1). Your code should set the data bank and direct page registers as needed. **HyperStudio neither allocates nor guarantees any direct page space for XCMD use;** it's the XCMD's responsibility to get it's own DP locations.

Table 1: The HyperStudio Inforamtion Block Structure

Offset	Size	Description
+0 - +1	WORD	ButtonID (the ID of the button pressed)
+2 - +3	WORD	CurrentCardID (the ID of the card with the button)
+4 - +7	LONG	handle to the script
+8 - +11	LONG	length of the script in bytes
+12 - +15	LONG	pointer to the command line text
+16 - +19	LONG	pointer to entry point for int. function handler

This table contains several important variables that you may work with in your XCMD. The most useful, however, are the last two: the long pointer to the command line text (so you may see what was passed to the XCMD), and the long pointer to the entry point for the HyperStudio internal function handler (so you may control features of HyperStudio from the XCMD) *Editor: This is analagous to calling an Applesoft ROM routinefrom assembly language or using a HyperCard "callback" on the Macintosh.*

Accessing the internal function handler is a matter of pushing any necessary parameters on the stack, selecting what you want HyperStudio to do by setting a value in the X register, and calling this entry point. The possible options are listed in Table 2. Only functions 5 and 8 (move to specific card and find text) require additional parameters to be passed on the stack: move to specific card needs the card ID (a word-sized value) pushed on the stack first, and find text requires a long pointer to the text to search for, and the find flags (another word-sized value; detailed below). Additionally, function 9 (set VOC flag) requires that you load the Y register with a boolean value to tell HyperStudio whether or not there's an Apple Video Overlay Card in the machine. (This allows you to override HyperStudio's control of the VOC.)

Let's take all that I've presented and apply it to a real, live, honest-to-goodness XCMD. I'll start with something simple, the Beep XCMD. This XCMD will simply beep the speaker once. It requires no input and does not control HyperStudio in any way.

Editor: All of the macro files called by the USE pseudo-op can be generated by running Macgen on the existing source. Every macro used is in the standard library. If one is not in your standard Merlin library, you may need to get an update. See the references at the end.

Listing 1: BEEP.XCMD

```
------------------
         lst   off    ;lose  playback
*--------------------------------------*
*                                      *
*BEEP xcmd by Eric Mueller    9-5-90 *
*copyright (c) 1990 Ariel Publishing *
*                                      *
*--------------------------------------*
*                                      *
* XCMD string: "BEEP"                  *
*                                      *
*--------------------------------------*
*                                      *
* This XCMD will execute a single      *
* _SysBeep call.                       *
*                                      *
*--------------------------------------*


         cas   in    ;case insensitive
         xc        ;allow 65c02 opcodes
         xc        ;allow 65816 opcodes
         mx    %00         ;16-bit M and X

         rel               ;relocatable code
         typ   $BC         ;type $BC (for an XCMD)
         lst   off
         use   beep.macs   ;use this macro file
         lst   rtn

**************************************************

         phb          ;usual startup
         phk
         plb

*=================================================
*
* Do the magic:
*

         _SysBeep          ;boink!

* Fix data bank and return to HyperStudio

         plb
         rtl   ;return to HyperStudio!

**************************************************

         lst   on
CHECKSUM chk
         lst   off

         sav   xcmd.beep.l ;make this link file
```

Table 2: HyperStudio Routine Numbers for the Internal Function Handler

value in X	function
1	move to first card in stack
2	move to last card in stack
3	move to previous card
4	move to next card
5	move to specified card (*)
6	redraw card (refresh screen)
7	(unused)
8	find text (*)
9	set video overlay card flag Y register = 0 means no VOC in the machine 1 means VOC in the machine

(*) these functions require additional parameters to be passed on the stack before the call is made.

```
------------------
```

To build and test this program, enter the listing into Merlin and then press apple-6. Merlin will assemble, link, and save the final XCMD.BEEP file on your drive. *(At the end of the assembly, a checksum value will be displayed to the left of the label "CHECKSUM". If it's not $E5, check your typing.)* If you're not using the Master XCMD system, change the last line of the program to "sav hs.xcmd.l" so that you'll get a resulting file entitled HS.XCMD. Copy this file into the same directory with HyperStudio.

Next, run HyperStudio and create a new button with the button text "Beep me, baby!" In the button actions dialog, click on the check box next to "Trigger an XCMD". You'll be presented with a dialog box containing a single edit line: if you're using the Master XCMD system, you'll need to type "BEEP", otherwise, simply make sure this line is blank. In either case, click on "Okay" to exit the dialog and then "Okay" one more time to finish creating the button.

Now, for the big test! You should have a button on the screen that says "Beep me, baby!" and the cursor should be the HyperStudio browse tool (the hand). Click on the button and listen: if the speaker beeps, your XCMD works as advertised!

Save your stack and go back into Merlin for this next example. It's a little more complex: called Beep2, it is a natural extension of the Beep XCMD. Beep2 allows you to specify, on the command line, how many times you wish the speaker to beep. This introduces the idea of grabbing parameters off of the command line and using them. I'll explain more after this listing...

Listing 2: BEEP2

```
------------------
         lst   off         ;lose the playback
*-------------------------------------------------*
*                                                 *
* BEEP2 xcmd       by Eric Mueller       9-5-90 *
*        copyright (c) 1990 Ariel Publishing      *
*                                                 *
*-------------------------------------------------*
*                                                 *
* XCMD string: "BEEP2 n"                          *
*                                                 *
*  n   number of times to beep speaker            *
*                                                 *
*-----------------------------------------------*
*                                               *
* XCMD will execute _SysBeep as many times as *
* you specify.                                *
*                                             *
*---------------------------------------------*

         cas   in          ;case insensitive

         xc                ;allow 65c02 opcodes
         xc                ;allow 65816 opcodes
         mx    %00         ;16-bit M and X

         rel                  ;relocatable code
         typ   $BC      ;type $BC (for an XCMD)
         lst   off
         use   beep2.macs ;use this macro file
         lst   rtn

*******************************************

tempPtr  equ   0,1,2,3  ;scratch pointer #1
tempPtr2 equ   4,5,6,7  ;scratch pointer #2
txtPtr   equ   8,9,a,b  ;long ptr to bttn's txt

*******************************************

         phb               ;usual startup
         phk
         plb

         MoveLong tempPtr;temp       ;save 0-3
         MoveLong tempPtr2;temp2     ;save 4-7
         MoveLong txtPtr;temp3 ;save 8-B

         sty   tempPtr ;save ptr to HyprStdio
         stx   tempPtr+2        ;...info block

         ldy   #12    ;get ptr to the command
         lda   [tempPtr],y ;line txt (@ offst
         sta   txtPtr  ;...+12 in info block)
         ldy   #14
         lda   [tempPtr],y
         sta   txtPtr+2

* Get length of the cmd line text & store it

         lda   #0         ;clear out high byte
         shortacc
         lda   [txtPtr] ;get len (txtPtr pts
                   ;to a P string which has
                    ;a leading length byte)
         longacc
         inc            ;(+1 because of the way
                               ;I do compares)
         sta   length   ;hold on to for later
         cmp   #1              ;no characters?
         beq   :beepLoop      ;yes - beep once

*=========================================
*
* Figure out how many times to beep:
```

```
*

:parse
        ldy   #0         ;start at first char
        shortacc
:loop
        iny
        cpy   length      ;at the end?
        beq   :stop       ;yes

        lda   [txtPtr],y
        cmp   #' '        ;space?
        beq   :loop       ;yes - ignore

                 ;no - copy them to a buffer
        ldx   #0
:loop2
        sta   :buffer,x
        inx
        lda   [txtPtr],y
        iny
        cpy   length
        bge   :stop    ;if we're at end, stop
        cmp   #' '            ;another space?
        bne   :loop2;no-keep getting int str
:stop
        longacc

        PushWord #0                   ;space
        PushLong #:buffer  ;push ptr to buff
        phx          ;push length of int. str
        PushWord #0       ;unsigned
        _Dec2Int

        pla              ;get the integer
        bcs   :stopBeep  ;if an error, leave

* Now that we know how many times to beep, do the
* big stuff here:

:beepLoop
        cmp   #0         ;is counter zero?
        beq   :stopBeep  ;yes - stop

        pha              ;hold counter
        _SysBeep         ;boink!
        pla              ;get counter back

        dec              ;bump it down
        bra   :beepLoop  ;and keep going

* We're done- restore things and return to HS

:stopBeep
        MoveLong temp;tempPtr ;restore 0-3
        MoveLong temp2;tempPtr2 ;restore 4-7
        MoveLong temp3;txtPtr ;restore 8-B

        plb
        rtl              ;return to HyperStudio!

*=================================================
*
* Data area
*

* Buffer for integer we get

:buffer  ds    20

* Storage for DP things we trash:

temp     ds    4
temp2    ds    4
temp3    ds    4

* Length of cmd. line text

length   dw    0

*************************************************

         lst   on
CHECKSUM chk
         lst   off

         sav   xcmd.beep2.l ;make this link file

------------------
```

To test this XCMD, build it in Merlin--the checksum value at the end should be $3D. (Remember to change the last line to "sav hs.xcmd.l" if you're not using the Master XCMD.)

Installing BEEP2

Now go back into HyperStudio, reload the stack you were working on earlier, and make a second button called "Beep me thrice, baby!" When you get a chance to enter the text to pass to the XCMD, type either "BEEP2 3" or just "3", again depending on whether or not you're using the Master XCMD.

Clicking on this button should cause the speaker to beep three times. You can continue to experiment with Beep2 by passing it different values... illegal strings (such as "-1") will cause nothing to happen, and no parameters will cause it to act like Beep and only beep once.

The Beep2 XCMD works by parsing out the command line (at the label ":parse") and looking for ASCII digits. Once I find the start of a number (in the ASCII range from '0' to '9'), I copy the number off to a buffer (called ":buffer") and stop when I reach the end of the command line or a space. (This routine can be used to pull several parameters off the stack, assuming they're separated by spaces.)

Finally, at the label ":stop", it's simply a matter of passing the number buffer to the toolcall _Dec2Int, which converts an ASCII string representing a value (such as "12") into a value (such as $C).

Dial an XCMD

The next XCMD I'm going to present is called Dial. It will talk to a modem plugged into your IIgs modem port and dial a phone number for you. This XCMD presents the concept of several flags on the command line to 'fine tune' performance. Based on these flags, the XCMD can dial in pulse or touch-tone, take the phone number from the command line or from highlighted text in the current text control, or even disconnect from the line after dialing a number!

This XCMD also takes control of the text tools to send the dialing string to the modem. XCMDs may do this as long as they preserve the environment (a similar limitation is imposed on new and classic desk accessories).

```
*------------------
         lst    off         ;lose the playback
*--------------------------------------------------*
*                                                  *
* DIAL xcmd          by Eric Mueller        9-5-90 *
*       copyright (c) 1990 Ariel Publishing        *
*                                                  *
*--------------------------------------------------*
*                                                  *
*XCMD string: "DIAL [+T|+P]|+H [n|+F]"             *
*                                                  *
* +T   force tone dialing                          *
* +P   force pulse dialing                         *
* +H   hang up (send just a return instead of      *
*     an ATDT string - use this after dialing to   *
*     make the modem disconnect from the line)     *
*                                                  *
*  n   phone number to dial                        *
* +F   dial phone number currently hilighted in a  *
*      text field                                  *
*                                                  *
*--------------------------------------------*
*                                            *
* This XCMD will dial phone numbers for you!*
*                                            *
* It assumes that you have standard Hayes-   *
* compatible modem attached to modem port.   *
*                                            *
*--------------------------------------------*


         cas   in          ;case insensitive

         xc               ;allow 65c02 opcodes
         xc               ;allow 65816 opcodes
         mx     %00        ;16-bit M and X

         rel              ;relocatable code
         typ    $BC    ;type $BC (for an XCMD)
         lst    off
         use    dial.macs  ;
         lst    rtn

********************************************

tempPtr  equ    0,1,2,3     ;scratch pointer #1
tempPtr2 equ    4,5,6,7     ;scratch pointer #2
txtPtr   equ    8,9,a,b     ;long ptr to
button's text

*********************************************

         phb                ;usual startup
         phk
         plb

         MoveLong tempPtr;temp     ;save 0-3
         MoveLong tempPtr2;temp2   ;save 4-7
         MoveLong txtPtr;temp3     ;save 8-B

         sty    tempPtr     ;ptr to HyperStudio
         stx    tempPtr+2   ;...info block

         ldy    #12         ;get ptr to command
         lda    [tempPtr],y;line text (@offset
         sta    txtPtr      ;+12 in info block)
         ldy    #14
         lda    [tempPtr],y
         sta    txtPtr+2

* Get the length of command line text & store

:getPtr
         lda    #0         ;clear out high byte
         shortacc
         lda    [txtPtr] ;get len (txtPtr pts
                    ;to a P string which has
                    ;a leading length byte)
         longacc
         inc            ;(+1 because of the way
                    ;I do compares)
         sta    length     ;hold on to this

*============================================
*
* Now we have a ptr to text attached to the
* btn (to pass to XCMD) in txtPtr. Let's
*  parse out the string:
*

:parseLine
         stz    hangUpFlag

         lda    #1     ;assume number starts at
         sta    numbOffset ;beg of dial string
```

```
* Did the user pick +T (tone dial) option?

         ldy    #1          ;start parsing at +1
         lda    #'T'
         jsr    findIt      ;find "+T"
         bcs    :noT        ;couldn't find it!

         sty    numbOffset;store offset to num
         lda    #'T'  ;make dialing pfx "ATDT"
         bra    :store ;and set it up

* Did user pick the +P (pulse dial) option?

:noT
         ldy    #1          ;start parsing at +1
         lda    #'P'
         jsr    findIt      ;find "+P"
         bcs    :noP        ;couldn't find it!

         sty    numbOffset;store offst to numb
         lda    #'P'    ;we got 'P', so set
         bra    :store  ;...the dialing option

* If neither, maybe we want to hang up?

:noP
         ldy    #1          ;start parsing at +1
         lda    #'H'
         jsr    findIt      ;find "+H"
         bcs    :cont       ;couldn't find it!

         inc    hangUpFlag
         brl    talkModem  ;we found it - do it now

* If a dialing option was used, store it away

:store
         shortacc           ;set dialing option
         sta    dialOpt
         lda    #4          ;set length of prefix...
         sta    dialString ;...to four ("ATDx")
         longacc            ;back to long acc
         bra    :getNumb

* We arrive here if no dialing option specified.
* Need to pull extra space out of the dialing pfx.

:cont
         shortacc
         lda    #3         ;since it's a P-string,...
         sta    dialString;...just bump the len
down
         longacc

* Now find the phone number to call:

:getNumb
         ldy    #1          ;start parsing at +1
         lda    #'F'
         jsr    findIt      ;find "+F" option
         bcc    :gotF       ;found it!
         brl    :getNumb1   ;couldn't find it;
use
                            ;number specified

* Since "+F" option used, phone number was
* hilited in text field. Grab it out & use it
* to dial.

:gotF
         PushLong #:startOffset ;ptr to buffer
         PushLong #:endOffset
         PushLong #0    ;use current TERecord
         _TEGetSelection
         bcc    :gotSel
         brl    :getNumb1 ;if error, leave now

:gotSel
         lda    :startOffset
         cmp    :endOffset ;the same?
         bne    :notTheSame ;no-selection made
         brl    :getNumb1   ;yes-no selection
                             ;so leave

:notTheSame
         PushLong #0      ;space
         PushWord #%11_101;allocate spc & give
                        ;text to us unformatted
         PushLong #:handle ;handle to text
         PushLong #0   ;bufferLength is ignored
         PushWord #0      ;styleRef is ignored
         PushLong #0   ;don't return style info
         PushLong #0     ;use default TERecord
         _TEGetText      ;get the text
         PullLong        ;ignore total length

         bcs    :getNumb1 ;if error, leave now

         MoveLong :handle;tempPtr2 ;deref
         lda    [tempPtr2]
         sta    tempPtr
         ldy    #2
         lda    [tempPtr2],y
         sta    tempPtr+2

* Now we know where selection is & we have
* in memory, copy out to the number buffer.

:moveSelection
         ldx    #0
         inc   :endOffset;(+1 for compare)
         shortacc
         ldy    :startOffset
:1
         lda    [tempPtr],y ;char of selection
         sta    number,x
         inx
         cpx    #31         ;buffer overflow?
         beq    :stop       ;yes - stop
```

```
        iny
        cpy   :endOffset ;all chars yet?
        bne   :1         ;no - keep going

:stop
        lda   #0
        sta   number,x  ;store nil @ end of
                  ;buffer (make a c-string)

        longacc       ;back to a long accum

        PushLong :handle  ;trash this memory
        _DisposeHandle

        bra   talkModem  ;and do dirty work!

:startOffset adrl 0  ;offset in curr TERecord
                      ;to the selection start
:endOffset adrl 0 ;offset in current TERecord
                  ;to the selection end
:handle  adrl  0          ;handle to all text

* If "+F" not used, copy phone number from
* command line into the buffer.

:getNumb1
        ldx   #0  ;offset into number buffer
        ldy   numbOffset ;get offset to #
        dey
        shortacc       ;down to short accum

:loop
        iny
        cpy   length  ;at end of the string?
        beq   :stop1     ;yes - stop

        lda   [txtPtr],y ;get a character
        cmp   #' '       ;space?
        beq   :loop      ;yes - keep looping

* note: remove next four lines if you want to
* allow non-digits into the dialing string

        cmp   #'0'       ;digit for number?
        blt   :loop      ;no - ignore it
        cmp   #'9'+1
        bge   :loop      ;no - ignore it

                  ;the character is okay!
        sta   number,x   ;store it in buffer
        inx          ;bump the offset forward
        cpx   #31       ;done 30 digits yet?
        bne   :loop      ;no - keep going

:stop1
        lda   #0        ;put a zero at end of
        sta   number,x ;the number buffer
        longacc        ;& go back to long acc

*=============================================
```

```
*
* The parsing is done and string is set up!
* Send it out to modem with the text tools.
*

talkModem

* Save old output device

         ~GetOutputDevice
         PullLong OldSlot0
         PullWord OldType0

* Save old output globals

         ~GetOutGlobals
         PullWord OutOr
         PullWord OutAnd

* Set the new output device and globals
* If you want to output to the printer change
the
* next line's parameters to "#1;#1" (use slot
#1).

:setOutDev
         ~SetOutputDevice #1;#2 ;Pascal type,
slot#

         ~SetOutGlobals #$7F;#$00

* Initialize it

         ~InitTextDev #1  ;initialize output

* Did we want to hang up (send just return)?

         lda   hangUpFlag
         bne   :hangUp    ;yes

* Send dialing pfx ("ATDT", "ATDP" or "ATD")

         ~WriteString #dialString ;send cmd

* Send out the phone number

         ~WriteCString #number ;send number

* Finally, drop carriage return behind it all

:hangUp
         ~WriteChar #$000D       ;send a C/R

* Reset the output device and globals

         ~SetOutputDevice OldType0;OldSlot0
         ~SetOutGlobals OutOr;OutAnd

* Re-initialize them
         ~InitTextDev #1  ;initialize output
```

```
*==========================================
*
* Finally, wrap stuff up and leave.
*
* Restore our zero page locations

         MoveLong temp;tempPtr   ;restore 0-3
         MoveLong temp2;tempPtr2 ;restore 4-7
         MoveLong temp3;txtPtr   ;restore 8-B

* Fix data bank and return to HyperStudio!

         plb
         rtl

*=========================================
*
* findIt: this subroutine will search through
* string ptd to by [txtPtr] (plus the offset
* in Y register) and look for character you
* specify in accumulator. returns with carry
*set if it couldn't find it, or carry clear &
* Y set to character after one you wanted,
* if it did find it.
*

findIt
         shortacc           ;short accumulator
         and    #$5f        ;make it uppercase
         sta    :findMe     ;find this char
:loop
         lda    [txtPtr],y
         cmp    #'+'        ;option?
         bne    :notYet     ;no
         iny
         cpy    length      ;at the end?
         beq    :stop       ;yes - stop now
         lda    [txtPtr],y ;no, check the next char
         and    #$5f        ;make it uppercase
         cmp    :findMe     ;character we're after?
         beq    :gotIt      ;yes! we got it!
:notYet
         iny                ;no - move to next char
         cpy    length      ;at end of the string?
         bne    :loop       ;no - keep going

:stop                       ;stop the search
         longacc            ;back to long accum
         sec                ;carry set = no match
         rts

:gotIt
         iny                ;bump Y forward one
         mx     %11         ;force longacc to assmble
         longacc
         clc
         rts                ;and leave

:findMe  db     0
```

```
*==========================================
*
* Data area: misc storage
*
* Length of button's text

length   dw    0

* Offset into str for phone number (past +T
or +P)

numbOffset dw  0

* Boolean: did we want to hang up or dial a
number?

hangUpFlag dw  0

* Storage for dial command and number

dialString dfb 4,'A','T','D'
dialOpt  asc   ' '         ;put "T" or "P"
number   ds    31          ;room for 30 digits
+
                           ;a trailing zero

* Storage for direct-page stuff we trample

temp     ds    4
temp2    ds    4
temp3    ds    4

* Storage for text tools stuff

OldSlotO adrl  0
OldTypeO ds    2
OldSlotI adrl  0
OldTypeI ds    2
InOr     ds    2
InAnd    ds    2
OutOr    ds    2
OutAnd   ds    2

******************************************

         lst   on
CHECKSUM chk
         lst   off

         sav   xcmd.dial.l ;make this file

------------------
```

After entering the listing, build it **(the checksum value should be $22)** and go into HyperStudio. Instead of giving you a specific button to create, I'm going to explain all of the different options for the Dial XCMD and let you decide how you wish you create the string. Note that, if you don't have a modem or you don't want to dial it, you can change the ~SetOutDevice line (at the label ":setOutDev" in the listing) to "~SetOutDevice #1;#1" in order to use slot one. This will send all of the program's dialing strings to the device plugged into your printer port.

The Dial program is very flexible; here's several examples of how to use it:

• `DIAL +T 1-913-469-6502`... dial the number 1-913-469-6502 in touch tone

• `DIAL +P 1-619-442-0522`... dial the number 1-619-442-0522 in pulse

• `DIAL 467-6429`... dial the number 467-6429 in your modem's default dialing mode

• `DIAL +t +f` ... dial the number highlighted in the current text field in touch tone

• `DIAL +h` ...hang up the modem after dialing (send a single CR)

The Tricks of the Trade

The source code has many interesting tricks, so let's take a look at it, section for section.

First, I get the pointer to the command line text (at the label ":getPtr") and then find the length of this text. The command line text is stored as a p-string, with a proceeding length byte, so it's easy to determine the length.

The next section of the code (starting at the label ":parseLine") works through the command line and looks for all flags. Flags can be found by looking for the proceeding "+" before the single-letter option. I look for the "+T" option, the "+P" option, and then the "+H" option, since these all come first on the command line (before the phone number or the "+F" flag). Both the "+T" and "+P" flags modify the dialing prefix (to either "ATDT" or "ATDP", respectively), while the "+H" flag immediately jumps to the section of code that talks to the modem.

I use the same subroutine, findIt, to look for each of the different flags. The findIt routine requires that you tell it at what offset to start looking on the command line (in the Y register), and what flag you're looking for (in the accumulator). It will return with the carry set if the flag couldn't be found, or with the carry clear and the Y register pointing to the character _following_ the flag, if it was found. The findIt routine is very handy, and you may find some use in your own XCMDs.

Once those flags are found, it's time to determine what phone number we're dialing (at the label ":getNumb"). First, I look for the "+F" flag. If that's found, I branch to the label ":gotF" and work with the current text field. This is a matter of two special TextEdit calls: _TEGetSelection, which returns the starting and ending offset of the selection in the current text record, and then _TEGetText, which actually gives me all of the text in the current record. Combining these two calls will let me pull the currently highlighted text out of the text field (this happens at the label ":moveSelection") and into a buffer in the XCMD, called "number".

If the "+F" flag was not used, however, it's a simple matter of copying the phone number from the command line right into the "number" buffer. This happens at the label ":getNumb1".

Finally, the flags and phone number are parsed out, and it's time to dial the phone, at the label "talkModem". I save the current state of the text tools by getting the output device and output globals. Then, I set the new text device and globals. Finally, it's just a matter of writing out the dialing prefix ("ATDT", "ATDP", or just "ATP"), the phone number, and a carriage return behind all of it.

From that point out, it's downhill: leaving the XCMD is simply a matter of restoring the text tools to their previous state, restoring the direct page locations, fixing the data bank, and RTLing back to HyperStudio.
Following that part of the listing is the "findIt" subroutine, which scans the command line in search of a target character. It's fairly well commented, and I won't go into details about it here, other than to say that it's worth studying and understanding. The final section of the listing is simply a data area.

The final sample XCMD is called Date. It will search through a stack and look in each text field for the current date (from the control panel setting). This allows you to set up a stack and put each day's events on each card, with the events labeled by a text field at top (such as "Appointments for 9/6/90"). You can also use this as a generic automated retrieval program, assuming that your data is stored in a text field and has each day's information labeled with the date.

Listing 3: Date.XCMD

```
-------------------
         lst    off         ;lose the playback
*---------------------------------------------------*
```

```
*                                               *
* DATE xcmd      by Eric Mueller        9-5-90 *
*      copyright (c) 1990 Ariel Publishing     *
*                                               *
*-----------------------------------------------*
*                                               *
* XCMD string: "DATE [+E|+R]"                   *
*                                               *
* +E   search only editable fields              *
* +R   search only read-only fields             *
*                                               *
*-----------------------------------------------*
*                                               *
*This XCMD will search through all text flds    *
*(with any exceptions you specify with +E or    *
* +R) and find the current date, & then jmp     *
* to that card.                                 *
*                                               *
* The actual date text searched for is from     *
* control panel. If the control panel is set    *
* to give you date in mm/dd/yy format (for      *
* example), & current date is September 6th,    *
* 1990, this XCMD searches for "9/6/90". (Note:*
* spaces are removed from date string prior to *
* executing the search, since the cntrl panel  *
* returns the date as " 9/ 6/90".)              *
*                                               *
*-----------------------------------------------*

         cas   in          ;case insensitive

         xc                ;allow 65c02 opcodes
         xc                ;allow 65816 opcodes
         mx    %00         ;16-bit M and X

         rel              ;relocatable code
         typ   $BC        ;type $BC (for an XCMD)
         lst   off
         use   date.macs  ;use this macro file
         lst   rtn

************************************************

ptrInfo  equ   0,1,2,3 ;long ptr to HS info blk
txtPtr   equ   4,5,6,7 ;long ptr to button's txt

************************************************

         phb               ;usual startup
         phk
         plb

         MoveLong ptrInfo;temp ;save 0-3
         MoveLong txtPtr;temp2 ;save 4-7

         sty   ptrInfo  ;save ptr to HyperStudio
         stx   ptrInfo+2 ;...information block

         ldy   #12        ;get ptr to the command
         lda   [ptrInfo],y ;line text (at offset
         sta   txtPtr     ;...+12 in info block)
         ldy   #14
         lda   [ptrInfo],y
         sta   txtPtr+2

* Get length of command line text and store it

         lda   #0           ;clear out high byte
         shortacc
         lda   [txtPtr] ;get len (txtPtr points
                        ;to a P string which has
                        ;a leading length byte)
         longacc
         inc            ;(+1 because of the way
                        ;I do compares)
         sta   length     ;hold on to this for
later

*==============================================
*
* Now we have a pointer to text attached to the
* bttn (to pass to XCMD) in txtPtr. Let's parse
* out the string:
*

:parse1
         lda   #%110    ;assume we can search
                        ;all fields, to start w/
         sta   findFlags

* Did user pick +E (editable fields only) opt?

         ldy   #0          ;start parsing at +0
         lda   #'E'
         jsr   findIt      ;find "+E"
         bcs   :noE        ;couldn't find it!

                           ;yes
         lda   findFlags
         and   #%101  ;turn off read-only bit
         sta   findFlags
         bra   :cont

*Did user pick +R (read-only flds only) opt?

:noE
         ldy   #0         ;start parsing at +0
         lda   #'R'
         jsr   findIt      ;find "+R"
         bcs   :cont       ;couldn't find it!

         lda   findFlags
         and   #%011   ;turn off editable bit
         sta   findFlags

* Now we have flags set up. Get date from the
* ctrl panl so we've something to search for.

:cont
        PushLong #dateBuffer ;buff for date
```

```
         _ReadAsciiTime ;get from the toolbox

                   ;date is first 8 bytes of
                   ;buffer, always "xx/xx/xx"

* Remove any spaces from the date string.
:killSpcs
         short            ;short registers
         ldx   #0
         ldy   #-1
:loop
         iny
         cpy   #9     ;done all characters?
         beq   :done      ;yes
         lda   dateBuffer,y ;get a character
         and   #$7f       ;clear high bit
         cmp   #' '       ;is it a space?
         beq   :loop      ;yes - ignore it!
         sta   dateBuffer,x ;no - put it back
         inx              ;bump storage index
         bra   :loop      ;and keep going

:done
         stx   dateLength ;store len date str
         long             ;back to long regs

*========================================
*
* Now date is set up as a p-string & so are
* the flags for the find command. Execute it.
*

* Get the addr of the HyperStudio entry point

:getEntry
         ldy   #16    ;get addr to the HS...
         lda   [ptrInfo],y ;...special fn
         sta   :JSLaddr   ;...handler (does
FIND)
         iny
         iny
         lda   [ptrInfo],y
         sta   :JSLaddr+2 ;store bank

* Set up the parameters for the call.

                   ;push a pointer to the buffer
         PushLong #dateLength
         PushWord findFlags

         ldx   #8         ;option #8 is FIND
TEXT

* Set up stack to call HyperStudio and then
return
* control to us.

:setReturn
         phk              ;push the return
address
         per   :rtnAddr-1 ;

         lda   :JSLaddr+2 ;push call addr
         shortacc
         pha              ;this pushes bank byte
         longacc
         lda   :JSLaddr   ;and this, the address
         dec              ;(again, -1 for stack)
         pha

* Now give control to HyperStudio for the find:

         rtl    ;call HyperStudio's routine
                          ;via RTLing to it

:rtnAddr                  ;HyperStudio returns
                          ;control to here

* If text was found, it moves the user to the
* card. Otherwise, a dialg ("No match found") is
* displayed. In either case, we're done.

:done
         MoveLong temp;ptrInfo ;restore 0-3
         MoveLong temp2;txtPtr ;restore 4-7

* Fix data bank and return to HyperStudio!

         plb
         rtl          ;return to HyperStudio!

:JSLaddr adrl  0   ;holds addr to func handler

*==========================================
*
* findIt: this subroutine will search through
* the string pointed to by [txtPtr] (plus offset
* in the Y register) & look for character you
* specify in the accumulator. returns with carry
* set if it couldn't find it, or carry clear and
* Y set to the character _after_ the one you
* wanted if it did find it.
*

findIt
         shortacc         ;short accumulator
         and   #$5f       ;make it uppercase
         sta   :findMe    ;find this char
:loop
         lda   [txtPtr],y
         cmp   #'+'       ;option?
         bne   :notYet    ;no
         iny
         cpy   length     ;at the end?
         beq   :stop      ;yes - stop now
         lda   [txtPtr],y ;no, ck next char
         and   #$5f       ;make it uppercase
         cmp   :findMe    ;char we're after?
         beq   :gotIt     ;yes! we got it!
:notYet
```

```
         iny                 ;no - next char
         cpy   length        ;at end of the str?
         bne   :loop         ;no - keep going

:stop                        ;stop the search
         longacc             ;back to long accum
         sec               ;carry set = no match
         rts

:gotIt
         iny                 ;bump Y forward one
         mx    %11         ;force longacc to asm
         longacc
         clc
         rts                 ;and leave

:findMe  db    0

*=============================================
*
* Data area: misc storage
*
* Length of command-line text

length   dw    0

* Flags for FIND command go here

findFlags dw   0

* Toolbox returns date/time in this buffer

dateLength db  0            ;length byte
                      ;(we fill this in later)
dateBuffer ds  20

* Storage for direct-page stuff we trample

temp     ds    4
temp2    ds    4

*************************************************

         lst   on
CHECKSUM chk
         lst   off

         sav   xcmd.date.l ;make this link file

------------------
```

(The checksum value for this code should be $AE.)

The Date command has two optional parameters: "+E", to search only editable text fields, and "+R", to search only read-only fields. I suggest taking the stack we've been working with and adding on several cards with text fields: "appointments for 10/1/90", "appointments for 10/2/90", and so on. If you'd like, make 31 cards, one for each day of the month!

Date Code

After the usual startup code (preserve our direct page space, get the pointer to the command line text, get length of the command line text), I start parsing out the command line (at the label ":parse1").

The first step is to set up the default find flags: I want to search both read-only and editable fields. However, if the user specifies either the "+E" (editable fields only) or the "+R" (read-only fields only) flag, I'll turn some bits off in this flag. (Note that bit zero, which controls whether or not the search is case-sensitive, is left at zero [case insensitive] for this XCMD.)

Parsing the flags works much the same as it did in the Dial XCMD--I look for the "+E" flag, then the "+R" flag, and turn off bits one or two in the find flags, if I find either.

Once I have the find flags set up based on the command line flags ("+E", "+R", etc), I'm ready to get the text to search for. Since I'm after the current date, I can ask the toolbox for the time and date, and it will give it to me in an ASCII form (at the label ":cont").

According to the Apple IIgs Toolbox Reference: Volume 1, page 14-16, _ReadAsciiTime always returns the date as the in eight characters of the buffer I specify, in the format "xx/xx/xx" (the month, day, and year positions can move around based on the user's control panel settings). However, if one of those three values (month, day, or year) are a single digit, they're padded with a single space to right-justify them within their field. What all this means is, the sixth day of September will be returned as " 9/ 6/90" (assuming the control panel format is mm/dd/yy) instead of "9/6/90". As a result, the next section of code (at the label ":killSpcs") 'collapses' the date by removing any spaces in the first eight characters.

Now that both parameters are set for the find call (I have the flags and the text to search for), it's time to return control to HyperStudio and execute the find. At the label ":getEntry", I get the address of the HyperStudio entry point, then push the parameters for the find call on the stack and load the X register with 8.

At the label ":setReturn", I push our return address on the stack and then HyperStudio's entry point on the stack, as well. Then, executing an RTL (right before the label ":rtnAddr") causes control to be passed back to HyperStudio. When HyperStudio RTLs back to my code, it will find my return address on the stack and come right back to the label ":rtnAddr". (These convolutions--pushing my return address, pushing HyperStudio's address, RTLing to it--are used in order to avoid self-modifying code. They actually aren't as confusing as they may seem. If you'd like, think of it as the inner working of a JSL.)

Finally, the XCMD is finished (at the label ":done") and the usual clean-up is done before returning control to HyperStudio again.

I hope these four examples have given you a small taste of what's possible with HyperStudio XCMDs. You've got the knowledge to go out and create your own XCMDs to do almost anything--they can work with add-on hardware, and they can still interface with HyperStudio. The power of HyperStudio is already extensive, and when you add XCMDs, there are almost no limits at all.

References:

GEnie, the General Electric Network for Information Exchange
401 N. Washington St.
Rockville, MD 20850
1-800-638-9636

CompuServe Information Services
5000 Arlington Centre Blvd.
Columbus, OH 43220
1-614-457-8600

Both GEnie and CompuServe have large libraries of HyperStudio stacks, as well as the Exerciser source code examples (in languages besides assembly) and other miscellaneous XCMDs.

Roger Wagner Publishing, Inc.
1050 Pioneer Way, suite 'P'
El Cajon, CA 92020
1-619-442-0522

You can order the HyperStudio XCMD Exerciser disk (with sample code in the five languages) for $10. You can also get the HyperStudio XCMD Library Disk Volume 1 (by Ken Kashmarek) for $49.95. The XCMD Library Disk includes the Master XCMD and a set of 19 new XCMDs for HyperStudio to experiment with.

ProDOS Power from Applesoft

by Jerry Kindall

Bo may know diddly, but Jerry knows Applesoft and BASIC.SYSTEM. So much so, in fact, that he was commissioned to produce MicroDot (a BASIC.SYSTEM replacement), for Kitchen Sink Software. Using MicroDot and/or the commands introduced in this article can really provide some power and flexibility to Applesoft disk routines. - Editor.

At the heart of ProDOS is the Machine Language Interface, or MLI. Through the MLI, ProDOS accepts all its commands. No command comes to ProDOS but through the MLI, you might say.

BASIC.SYSTEM has as its primary task translating Applesoft programs' disk commands into MLI-speak. It's possible to bypass BASIC.SYSTEM and issue commands directly to the MLI from an Applesoft program, and it only requires a few bytes of machine language code to do it. Along the way, I'll share some things I've learned about BASIC.SYSTEM and stuff like that. I'll do it all in a cookbook format (more or less) so you can use these routines without having to know (or care) how they work.

The MLI Caller

Let's develop a set of Applesoft subroutines to do MLI work for us. The first thing we need is an initialization subroutine that POKEs in the short machine language routine we'll use to call the MLI. Here's what the MLI caller looks like in pidgin assembler:

```
jsr $BF00      ;call the MLI
$00            ;MLI command goes here
$0000          ;addr of parmlist goes here
sta $FF        ;store error code in loc 255
rts            ;back to BASIC
```

This 9-byte machine language thing *(Editor: Oh Jerry, I love it when you talk techie)* is completely relocatable. Our setup routine will look at the variable MLI to find out where it should POKE the machine language code -- if MLI is zero, we'll assume a default value of 768, the beginning of page 3.

```
1000  REM [ Applesoft MLI Routines ]
1010  REM  -----------------------
1020  REM Initialization and setup
1030  REM Install MLI Caller ML routine
1040  IF MLI = 0 THEN MLI = 768
1050  POKE MLI,32: POKE MLI+1,0: POKE
MLI+2,191
1060  POKE MLI+3,0: POKE MLI+4,0: POKE
MLI+5,0
1070  POKE MLI+6,133: POKE MLI+7,255: POKE
MLI+8,96
1080  DEF FN PK (X) = PEEK (X) + PEEK
(X+1) * 256
1090  DEF FN HI (X) = INT (X/256)
1100  DEF FN LO (X) = X - FN HI (X) * 256
1110  RETURN
```

Here we also set up some user-defined functions for dealing with two-byte PEEKs and POKEs. FN PK returns the two byte value at the address specified. FN HI and FN LO return the high and low bytes of the value specified, just what you need when you're getting ready to POKE the address of something into a pointer somewhere. If you're not using user-defined functions in your Applesoft programs for formulas you use over and over, you're missing out.

Now we need another bit of BASIC "glue" to handle the actual MLI call.

```
1120  REM Call the MLI please
```

```
1130  POKE MLI+3,CMD: POKE
MLI+4,FN LO       (PARM): POKE
MLI+5, FN   HI (PARM)
1140  CALL MLI: ERR = PEEK
      (255): RETURN
```

This code assumes CMD and PARM have already been set up with the MLI command number and address of the parameter list, and returns the MLI result code in ERR (zero means no error).

Open Sesame

One of the fundamental things you want to be able to do with a ProDOS MLI call is to open a file for reading and writing. Doing this with a ProDOS MLI call is a little bit silly, though, because BASIC.SYSTEM already does that for us just fine, and it moves our variables down and allocates us a buffer, as well (which we'd have to take care of ourselves if we called the MLI directly). We'd also have to somehow get the pathname of the file into the format that ProDOS expects, which we'll get to eventually but not right now.

But doing it the BASIC.SYSTEM way also has a couple of drawbacks. First, you have to know in advance the type of the file you're opening. This isn't a problem because we'll write a routine in a second to find out a file's type, but the ProDOS MLI doesn't actually care. In fact, BASIC.SYSTEM has to do a GetFileInfo call to know whether or not you specified the right filetype for the file, and we'll do another one to find out the right filetype so we can fake BASIC.SYSTEM out.

Second, and more importantly, since BASIC.SYSTEM assumes you're opening a file on which you'll be doing reads and writes of regular text data, it sets the ProDOS newline mode to look for the carriage return when reading. What this means is when you read from a file using MLI commands, you might get all the data you asked for, unless there's a carriage return somewhere in the data, in which case you'll get all the data up to and including the carriage return. We want to turn this mode off so we can do regular binary I/O on the file.

```
1150 REM Open a file and turn off Newline
1160 POKE 48851,0: PRINT CHR$(4);"OPEN
";F$;",T";T$: POKE 48851,127
1170 REF = PEEK (48848): RETURN
```

Figure 1: Global Page Parameter List List

Hex	Dec	MLI call it's used with
$BEA0	48800	Create
$BEAC	48812	GetPrefix, SetPrefix, Destroy (delete)
$BEAF	48815	Rename
$BEB4	48820	GetFileInfo, SetFileInfo
$BEC6	48838	Online, SeMark, GetMark, SetEof, GetEof, SetBuf, GetBuf, Quit
$BECB	48843	Open
$BED1	48849	SetNewline
$BED5	48853	Read, Write
$BEDD	48861	Close, Flush

To use this routine, set F$ to the name of the file to open, and T$ to the type of the file. (You can use hex numbers like "$FF", decimal numbers like "255", or mnemonics like "SYS", but they've got to be in a string.) The routine will return the reference number of the file BASIC.SYSTEM just opened in REF. (ProDOS assigns this number; you have no say in it.) You will need this number later to access the file; the MLI knows files by their number, not their name.

What was I PEEKing and POKEing there? BASIC.SYSTEM has a few MLI parameter lists in its global page (which begins at $BE00), and I'm simply borrowing the SetNewline parameter list. In case you're not familiar with MLI parameter lists, each time you call the MLI you set up an area of memory containing the parameters of the command you want to execute. Then you tell the MLI where this parameter list is and it looks there for the information it needs. Since BASIC.SYSTEM calls the MLI, it has a number of MLI parameter lists, and the folks at Apple were good enough to document the addresses of these parameter lists (c.f. Figure 1)

As you can see, many of the lists are used with more than one type of MLI call, since the calls in question require the same number of bytes for the parameter list, or have parameters in common, or whatever. The PEEK (48848) grabs the reference number of the file just opened from the Open parameter list -- the referece number is returned at relative byte +5 of the Open parmlist. The POKEs to 48851 set the mask byte for the SetNewLine parameter list to zero, then change it back to its usual value ($7F). It's often important to put things back the way you found them when using

BASIC.SYSTEM's parameter lists, because, in some cases, BASIC.SYSTEM depends on them being set properly to operate correctly. All manner of evil might happen if you mess up. Obviously you don't have to be quite as careful with the areas which are used for multiple functions, as BASIC.SYSTEM sets these up as necessary for each call.

A complete breakdown of what each type of parameter list contains is beyond the scope of this article. Both Apple's ProDOS 8 reference manual and *Beneath Apple ProDOS* include this information, and any serious ProDOS hacker should have at least one of these two books. If you have neither, get one ASAP.

I've Got Your Number

Now that we've got the file open and know its reference number, we can now do all sorts of obscene things to it that BASIC.SYSTEM would not normally allow. Let's find out how long it is with a GetEof call. ("Eof" is a Martian word meaning "End of file".)

```
1180  REM Get file length
1190  POKE 48839,REF: CMD = 209: PARM =
      48838: GOSUB 1130
1200  LN = FN PK (48840) + PEEK (48842) *
      65536: RETURN
```

This little routine tells you the file's length in bytes. We POKE the reference number we got from the OPEN routine into the parameter list, then do the call. After the call, the length of the file is stored in three bytes, in relative bytes +2 through +4 of the parameter list, in the usual backwards ProDOS order. (ProDOS needs three bytes because a file can be up to 16 megabytes in length.) We might want to check ERR to make sure the call was successful, but if the file opened without error, there's no reason for the GetEof call to fail.

Why might we want to do this, anyway? Well, suppose we were writing a file copy routine in BASIC. We might just need to know how long the file is in bytes so we know how many passes are required to copy the entire file with the memory buffer we have available. Or maybe we want to check that a particular BIN file is 8192 bytes long (a hi-res picture, doncha know) before loading it.

One cheap trick we could pull is to simply close the file. BASIC.SYSTEM allocated us a 1K buffer for the file. We can PEEK this buffer's address out of the Open parmlist. If we close the file through the MLI, BASIC.SYSTEM won't know about it and will keep that memory reserved. Then we can use that memory for something else. There are much better ways to reserve memory under BASIC.SYSTEM, which is why I called this a cheap trick. There are other drawbacks too, so I don't reccommend it, but it's kind of fun to contemplate, isn't it?

The Two Rs

We can, of course, read from and write to our open file. This would be, essentially, the same as a BLOAD. We can even write a routine to move the position-in-file pointer to an arbitrary location, giving us random access BLOAD and BSAVE capabilities. We already have random-access BLOAD and BSAVE capabilities with the B parameter, though, so what's the advantage? No particular advantage, until you start performing multiple random accesses to the same file. BLOAD and BSAVE open the file each time such a read or write is performed. With our MLI caller, we open the file once. An OPEN call adds quite a bit of overhead to BLOAD and BSAVE, which we save by calling the MLI directly.

A good example of a program in which this would be useful is John Link's nifty SuperPatch, an AppleWorks patch program. As part of its operation, SuperPatch checks out AppleWorks patch locations and reports to the user whether or not each patch is installed. SuperPatch uses BLOAD and the B parameter to perform this task. By using the MLI caller, we could speed this up significantly. Let's write some routines to let us do random-access BLOADs and BSAVEs.

```
1210  REM Set position-in-file to BYTE
      (changes variable BYTE)
1220  POKE 48839,REF: POKE
      48842,BYTE/65536: BYTE = BYTE-
      PEEK(48842)*65536
1230  POKE 48841,FN HI (BYTE): POKE
      48840,FN LO (BYTE)
1240  CMD = 206: PARM = 48838: GOSUB 1130:
      RETURN
```

The calculations in the above routine look complicated because we have to handle not only the high byte and the low byte of the position-in-file pointer, but also a middle byte. ProDOS supports files of up to 16 megabytes in length, so three bytes are needed to specify a byte position within the file. I handled this in my routine by POKEing the highest byte first and adjusting the variable BYTE to "strip out" the highest byte. Then I used the regular FN LO and FN HI on what was left over. I do a SetMark MLI call once the parm list is set up properly.

```
1250   REM Set position then fall through
       to Read
1260   GOSUB 1220
1270   REM Read NUM bytes from file into
       memory at ADR
1280   POKE 48854,REF: POKE 48855,FN LO
       (ADR): POKE 48856,FN HI (ADR)
1290   POKE 48857,FN LO (NUM): POKE 48858,
       FN HI (NUM)
1300   CMD = 202: PARM = 48853: GOSUB 1130
1310   NUM = FN PK (48859): RETURN
```

The Read routines above are fairly straightforward. For convenience, I provided two entry points. If you enter at 1260, the SetMark routine is automatically called first, allowing you to simulate a random-access BLOAD by setting NUM, ADR, and BYTE appropriately and calling one entry point. The second entry point (1280) just starts reading from wherever the position pointer happens to be. NUM is modified at the end of the routine to be the number of bytes actually read, which may be less than than requested if there wasn't enough data left in the file to fulfill the request. ProDOS does not consider this an error condition, so you should check NUM to find out if you got all the data you asked for.

```
1320   REM Set position then fall through
       to Write
1330   GOSUB 1220
1340   REM Write NUM bytes from memory at
       ADR into file
1350   POKE 48854,REF: POKE 48855,FN LO
       (ADR): POKE 48856,FN HI (ADR)
1360   POKE 48857,FN LO (NUM): POKE
       48858,FN HI (NUM)
1370   CMD = 203: PARM = 48853: GOSUB 1130:
       RETURN
```

The write routines are almost identical, with two entry points, one to simulate a random-access BSAVE and another to simply perform a bulk memory write to the current position-in-file pointer. By the way, with all three of these routines you should be checking the ERR variable on return to make sure no errors ocurred.

Relativistic Limits

I'm running out of time and space for this article, but I promised you a routine which would return a file's type for use with the OPEN routine. For this task, we need to use the ProDOS GetFileInfo call. As it turns out, BASIC.SYSTEM already has a command for doing a GetFileInfo call. It's called VERIFY. Of course, usually this command is used only to verify that a particular file exists, but it actually returns a wealth of valuable information in BASIC.SYSTEM's GetFileInfo parm list. For now, we'll just PEEK out the file type.

```
1380   REM Get file type into T$
1390   PRINT CHR$(4);"VERIFY ";F$: T$ =
       STR$ (PEEK (48824)): RETURN
```

Let's also add:

```
1144   REM Get file type and fall thru to
       Open
1145   GOSUB 1390
```

This gives us one entry point which will get the file's type and open it for us, just like we did with the read and write routines. It's not on an even line number anymore; guess I wasn't thinking ahead enough.

Since I ran out of spacetime for this article, I'll have to write a sequel. In the next episode, I'll show you how to do a number of things, including PEEKing out all the GetFileInfo info (and deciphering the date format), changing a file's info with SetFileInfo, storing a ProDOS string in memory without using POKE (which we'll need for SetFileInfo), a couple of nifty tricks with GetMark/SetMark/GetEOF/SetEOF, reading a bunch of info from the ProDOS and BASIC.SYSTEM global pages, listing all online volumes, and much, much more. In fact, there are so many fun things you can do with BASIC.SYSTEM and ProDOS that I might even need a third article to cover them all.

See you then!

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.

Inner*Express* $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

OverScan

by David Kopper

What is overscan? It is the process of 'drawing' a set of lines in the screen border. Originally, I thought this was an impossible task. The ACS demo from the French FTA group changed my mind, however. *[The FTA group released a very impressive disk that had a small overscan demonstration. -ed]* My next thought was that it would be difficult to program (i.e. lots of ugly interrupt service routines). Then, I thought that even if it were possible and not too ugly, it couldn't be done without breaking all the rules.

Everyone I spoke with seemed to come to the same general conclusions. This article is the result of my attempt to see if you really can 'draw' in the screen border.

How to Overscan

The 'trick' to overscan is that you change the screen border color at exactly where you want the line 'drawn' on the screen border. (Because of this, you can only effectively 'draw' horizontal lines in a single color in the border.) There are several problems that I had to solve to be able to do this. The first is, how can I change the screen border color?

Unfortunately, there are no tool calls that can be used to change the border color. I have to change it by storing the new color in the screen color register at location $C034 (using a read, modify and write sequence of instructions, in order to preserve the bits that I am not modifying). The screen border color is the low four bits, so we'll preserve the high four bits of the byte.

The next question is, how can I change the screen border color at exactly the right time?

Super Hi-Res

Before I answer that, I need to explain some details about the Super Hi-Res (SHR) graphics screen and interrupt processing.

The Video Graphics Controller (VGC) refreshes the video screen 60 times per second. Each refresh cycle starts at the top of the screen and redraws the entire screen, one line at a time (including the screen border).

The SHR screen has some additions to it that make it more interesting and more difficult to program than the other display modes. Each row of pixels on the SHR screen is referred to as a scan line. Each scan line has its own control byte, appropriately named Scan line Control Byte, or SCB. The SCB has the following structure:

Bit 7: 320- or 640-pixel mode
Bit 6: Generate a scan line interrupt or not
Bit 5: Enable fill mode or not
Bit 4: Reserved
Bits 3 to 0: Palette number for this scan line

The bit that grabbed my interest was bit six. If I set bit six (generate scan line interrupt) to a one, and enable scan line interrupts, then the VGC will interrupt the CPU exactly when the scan line is about to be drawn.

The second problem has now evolved into: how do I process a scan line interrupt? Or, in more general terms, how does the IIgs process interrupts?

Interrupt Processing

An interrupt is a distraction that once you do something about it, it goes away. The same definition applies to your IIgs.

Interrupts occur whenever something happens that the CPU has to do something about. Your mouse, the serial ports on the back of your IIgs, pressing CTRL-APPLE-ESC, the vertical blanking interrupt of the SHR screen, and many other things cause interrupts.

How does the IIgs handle an interrupt? When an interrupt occurs, the CPU stops what is currently is doing and saves enough information to resume where it stopped. The next step is to find out what caused the interrupt and to do whatever is necessary to process it. The interrupt manager is in charge of doing this. The interrupt manager calls each of its interrupt handling routines/vectors in order of priority (this how serial port interrupts are handled before mouse interrupts). Once an appropriate handler is found for the interrupt, it is called, the interrupt is handled, and the interrupt manager returns control to the CPU.

The interrupt handler that we're interested in is the scan line interrupt handler. By setting up our own scan line interrupt handler, we can change the screen border color at exactly the right time.

The Code

Remember, this demo is primitive! It was constructed just to show how to get overscan working using toolbox calls. I developed this demo using the ORCA/M assembler with the APW interface files for System Disk 5.0 on my ROM 01 IIgs, using System Disk 5.0.2. I have also run this demo on a ROM 03 machine.

The Main, Startup_Tools and Shutdown_Tools routines are just glue to get the demo running. The only additional comment I'll add is that the memory attributes passed to _NewHandle are intentionally coded to be a sum of attributes. The reason is that if I ever port a program from my IIgs to some other computer, I would never remember the significance of $C015. The addition of all the attributes might remind me of what I was trying to get _NewHandle to allocate.

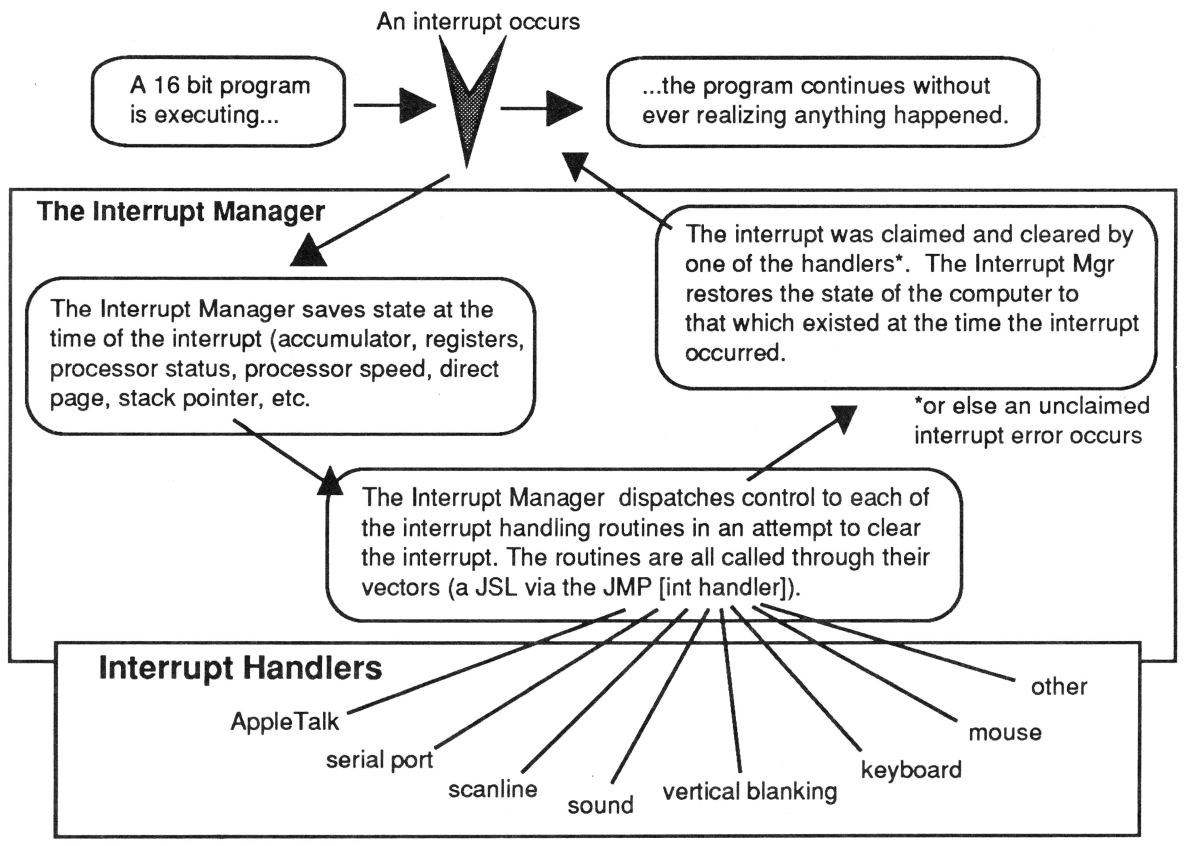
Startup_Demo is the routine that really does most of the hard work. Its goal is to setup my scan line interrupt handler and enable scan line interrupts on the two scan lines where I will change the border color.

Shutdown_Demo is the opposite of Startup_Demo. Its main purpose is to restore everything back to the way it was when this demo was started. This includes resetting the two scan lines, restoring the system scan line interrupt handler and restoring the original screen border color.

The EventLoop routine is my simple event loop which will simply wait for a key to be pressed. In your program, you'd probably use _GetNextEvent instead of checking the keyboard strobe directly.

ScanLineIntHndr is my scan line interrupt handler.

Figure 1: An Interrupt Handling Flowchart


An interrupt occurs
A 16 bit program is executing...
...the program continues without ever realizing anything happened.
The Interrupt Manager
The Interrupt Manager saves state at the time of the interrupt (accumulator, registers, processor status, processor speed, direct page, stack pointer, etc.
The interrupt was claimed and cleared by one of the handlers*. The Interrupt Mgr restores the state of the computer to that which existed at the time the interrupt occurred.
*or else an unclaimed interrupt error occurs
The Interrupt Manager dispatches control to each of the interrupt handling routines in an attempt to clear the interrupt. The routines are all called through their vectors (a JSL via the JMP [int handler]).
Interrupt Handlers
AppleTalk
serial port
scanline
sound
vertical blanking
keyboard
mouse
other

It is called by the interrupt manager with both the accumulator and x/y index registers set to be 8 bits wide. This routine sets the border color to the next border color in the border color table, then clears the interrupt and returns. The scan-line interrupt is cleared when bit 5 of the VGC interrupt clear register is set to zero.

Notes and Warnings

Overscan will only work while the SHR screen is being displayed. Scan-line interrupts do not occur in any other graphics mode.

While 'drawing' in the screen border with overscan, the mouse cursor can NOT be shown. There are two reasons for this restriction: first, the mouse cursor is drawn by the tool box during the processing of scan line interrupts. Since I replaced the scan line interrupt handler vector with my own, the tool box won't get the chance to draw the cursor.

The other reason is that the mouse cursor is drawn one scan line at a time. The tool box has to process as many scan line interrupts as the cursor is scan lines high. Since the scan-line interrupt routine doesn't expect the 'extra' interrupts, the border colors would probably not be what you expected.

An anomaly I've found is that if you enter the Classic Desk Accessory menu between the two scan-lines, then the screen border colors will flip from black with a blue line to blue with a black line. This is because the border color index is not maintained very well (it simply flips the border color from one to the next, not paying attention to which scan-line interrupt is being processed).

Enhancement Ideas

Setting up a SHR title screen with overscan lines joining part of the SHR picture would make a great INIT segment of an application. I can even imagine that games might want to use overscan to expand the screen area they 'draw' on (such as a racing game where the border color height shows how much of the race course has been completed).

You could reset the border color index on each vertical blanking by setting up a either a heartbeat task or add a task to the run queue that runs ever 1/60th of a second (see Miscellaneous tool call _SetHeartBeat or the new Desk Manager tool call _AddToRunQ).

OverScan Listing:

```
;************************************************
;
; OverScan
;
; Description: This program is just a demo that
;              will 'draw' a line in the screen
;              border area.
;
;              The code presented below will
;              'draw' a line in the screen
;              border.  However, the following
;              are the conditions to avoid while
;              such 'drawing' is occuring:
;
;              * Do NOT enable the mouse cursor
;                (the cursor is drawn during scan
;                line interrupts - and since we
;                don't call the system scan line
;                interrupt handler, the cursor
;                would not appear.  Even worse
;                is that the border color would
;                be changing when we didn't want
;                it do (based on the mouse
;                position)).
;
;              * Do not disable interrupts.
;
; Notice: Error handling in this demo is VERY
;         primitive.
;
; Copyright (c) 1990 Ariel Publishing. Some
; rights reserved.
;
;************************************************


; Read all the required macros/definitions...

        mcopy os.macros

        copy  2/ainclude/e16.memory
        copy  2/ainclude/e16.misctool
        copy  2/ainclude/e16.quickdraw


; Define the scan lines we will change the border
;  color on:

ScanLine1 gequ 100
ScanLine2 gequ 110

; Define the size of the stack this demo will
;  create.

StackSize gequ $1800

; Define the 'special' locations we'll be using:
```

```
KeyBoard    gequ $e0c000 ;Keyboard character
KeyStrobe   gequ $e0c010 ;Keyboard strobe
BorderColor gequ $e0c034 ;Screen backgrnd color
VGCIntReg   gequ $e0c023 ;VGC Interrupt registr
VGCIntClr   gequ $e0c032 ;VGC Interrupt Clear
;                        ; register
IntFlag     gequ $e0c046 ;Interrupt Flag -bit 3
;                        ; is set during a VBL
;                        ; interrupt


; Finally, the direct page variable:

HandleLoc gequ 0   ; Used to deref a handle


;******************************************
;
Main            Start
;
; Description: The main rtn - it will simply
; call subrouttines in the right order.
;
;******************************************

         using Data

; Setup the data bank

         phk
         plb

; Start the tools then Initialize demo...

         jsr   Startup_Tools
         jsr   Startup_Demo

; Draw a rect to make things more impressive

         PushLong #TheRect ;long ptr to rect
         PushLong #PenPatt ;long ptr to pattern
         _FillRect

; Wait for a key to be pressed

         jsr   EventLoop

; We're done, shutdown the demo and tools

         jsr   Shutdown_Demo
         jsr   Shutdown_Tools

         _QuitGS QuitParms
         rtl
         brk   $00

         end                               ;
Main
;********************************************
```

```
;
Startup_Tools  start
;
; Description: Load and start the needed tools.
;
;***********************************************

         using Data

; Start the tool locator, misc. tools, and the
;  memory manager.


         _TLStartUp
         bcs   error

         _MTStartUp
         bcs   error

         WordResult
         _MMStartUp
         bcs   error
         PullWord ProgramID

; Create a privite user ID for this demo

         lda   ProgramID
         clc
         adc   #$0100
         sta   PrivateID

; Save whatever is in HandleLoc for a bit...

         lda   HandleLoc
         sta   HandleTemp
         lda   HandleLoc+2
         sta   HandleTemp+2

; Allocate the direct page space for all of
;  the tools we'll be using:
;   three pages for quickdraw II

         LongResult
         PushLong #$0300
         PushWord ProgramID
         PushWord
#attrLocked+attrFixed+attrNoCross+attrPage+
attrBank
         PushLong #0
         _NewHandle
         bcs   error
         PullLong HandleLoc

; Start quickdraw II

         lda   [HandleLoc]
         pha
         PushWord #0         ;320 mode
         PushWord #0         ;def screen width
         PushWord ProgramID  ;User ID
```

```
        _QDStartup
        bcs   error

; Restore HandleLoc to its previous values...

        lda   HandleTemp
        sta   HandleLoc
        lda   HandleTemp+2
        sta   HandleLoc+2


        rts

error   anop
        brk $01

HandleTemp ds 4
        end                         ;
Startup_Tools


;******************************************
;
Startup_Demo   start
;
; Description: Perform application specific
;            initialization.  This demo will
;            save the system interrupt mgr,
;            setup its own interrupt manager
;            turn scan line interrupts on &
;            set scan line interrupt bit on 2
;            scan lines (in the SCBs).
;
; Assumption:  Interrupts are enabled!
;
;******************************************

        using Data

; Save the system scan line interrupt handler
;  for later restoration.

        LongResult
        PushWord #intrptMgr
        _GetVector
        bcs   error
        PullLong SystemIntMgr

; Install the jump to the system interrupt
;  manager.

        jsr   IMSetup

; Set the scan line interrupt vector to pt to
;  my scan line interrupt handler.

        PushWord #intrptMgr
        PushLong #MyIntMgr
        _SetVector
        bcs   error
```

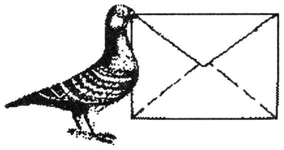
```
; Set the scan line interrupt bit on the two
;  lines where we will change the screen
;  background color.

scan_ints_on anop

         PushWord #ScanLine1
         WordResult
         PushWord #ScanLine1
         _GetSCB
         bcs   error
         pla
         ora   #scbInterrupt
         pha
         _SetSCB
         bcs   error

         PushWord #ScanLine2
         WordResult
         PushWord #ScanLine2
         _GetSCB
         bcs   error
         pla
         ora   #scbInterrupt
         pha
         _SetSCB
         bcs   error


; Everything is setup - return to our caller

         rts

error    brk   $02
         end                         ;
Startup_Demo


;*********************************************
;
Shutdown_Tools start
;
; Description: Shutdown the tools we used.
;
;*********************************************

         using Data

         _QDShutDown

         PushWord PrivateID
         _DisposeAll

         PushWord ProgramID
         _MMShutDown

         _MTShutDown

         _TLShutDown
         rts

         end                            ;
Shutdown_Tools


;*******************************************
;
Shutdown_Demo   start
;
; Description: Shutdown facilities that this
;         demo setup.  This involves
;         turning off the scan line
;         interrupt bits in two SCBs,
;         turning off scan line interrupts
;         (but only if they were not enabled
;         when this program started),
;         restore the system scan line
;         interrupt vector and finally,
;         restore the screen border color.
;
;*******************************************

         using Data

; Reset scan line interrupt bit of the scan
;  line control bytes on lines where we were
;  changing the border color.

         PushWord #ScanLine1
         WordResult
         PushWord #ScanLine1
         _GetSCB
         bcs   error
         pla
         and   #($ffff-scbInterrupt)
         pha
         _SetSCB
         bcs   error

         PushWord #ScanLine2
         WordResult
         PushWord #ScanLine2
         _GetSCB
         bcs   error
         pla
         and   #($ffff-scbInterrupt)
         pha
         _SetSCB
         bcs   error

; Restore the system interrupt manager.

         PushWord #intrptMgr
         PushLong SystemIntMgr
         _SetVector
         bcs   error

; Restore the screen border color, getting the
;  screen border color from battery backed up
;  ram.
;
; Note: The BorderColor location contains the
;       border color in the low four bits and
;       part of the real time clock in the high
;       four bits.

         WordResult
         PushWord #dspBrdColor
         _ReadBParam
         bcs   error
         PullWord OldBorderCl

         longa off
         sep   #$20
         lda   >BorderColor
         and   #$f0
         clc
         adc   OldBorderCl
         sta   >BorderColor
         longa on
         rep   #$20

; The border color demo has been shutdown
;  - return to our caller

         rts

error    brk   $03

         end                       ; Shutdown_Demo


;**************************************************
;
EventLoop      start
;
; Description: This is the eventloop for this
;              demo.  All we're going to do is
;              watch for a key press - once we
;              see one, we'll clear the keyboard
;              strobe and return to our caller.
;
;**************************************************

; Now check to see if there is a key.

loop     lda   >KeyBoard
         bpl   loop
         sta   >KeyStrobe
         rts
         end                           ; EventLoop


;*********************************************
;
MyIntMgr start
;
; Description: This is my Interrupt Manager.
```

```
;
;           We are replacing the regular
;           interrupt manager, because it
;           isn't fast enough for us.
;
;           This interrupt manager only
;           processes scan line interrupts
;           & vert blanking interrupts.
;           Any other interrupt is passed to
;           the original interrupt manager.
;
;           The border color will change
;           during the processing of a scan
;           line interrupt.
;
; Assumption:  We are in native mode with the
;              return address/processor status
;              pushed on the stack.
;
;*******************************************

         using Data

         clc
         xce
         sep #$20
         php
         pha
         phx
         longa off

; Check to see if interrupt is a scan line
;  interrupt.
;
; Note: If bit 5 of the VGCIntReg is set then
;       this interrupt is for a scan line.

         lda   >VGCIntReg
         and   #$20
         beq   VBLCheck


; Delay 'till VGC gets to the beginning of
;  the scan line (Scan line interrupts occur
;  at the right end of the screen).

         ldx   #$12
delay    dex
         bne   delay


; Set the screen border color to the next color
;  in the border color table.
;

         lda   >ColorIndex
         tax
         lda   >BorderColor
         and   #$f0
         clc
         adc   >ColorTable,x
         sta   >BorderColor
         lda   >NextColor,x
         sta   >ColorIndex

; Clear the scan line interrupt by clearing
;  the scan line interrupt bit in the VGC
;  interrupt clear register.
;
; Note: Clearing bit 5 of the VGCIntClr
register
;       will clear the interrupt from the VGC.

         lda   >VGCIntClr
         and   #($ff-$20)
         sta   >VGCIntClr

; Since we are the interrupt manager, the way
to
;  return from this interrupt is to pop all the
;  saved information from the stack and RTI
back
;  to the interrupted program.

         clc
         plx
         pla
         plp
         rti


; Well, the interrupt wasn't caused by a scan
;  line.  Lets check to see if it was caused by
;  a vertical blanking interrupt.
;
; Note: If bit 3 of IntFlag is set then
;       this interrupt is for a Vertical
;       Blanking.

VBLCheck anop

         lda   >IntFlag
         and   #$08
         beq   SystemInt

; Set ColorIndex to zero - this will cause the
;  scan line interrupts to change the border
;  colors in the same sequence for each time
;  the screen is refreshed.

         lda   #0
         sta   ColorIndex

; Continue processing of the VBL interrupt
;  in the system interrupt manager.

;        bra   SystemInt


;Interrupt is not a scan line interrupt, but
```

```
;it may have been a Vertical BLanking
;interrupt.  We really don't care, lets just
;pass this interrupt onto system interrupt
;manager.

SystemInt anop

         plx
         pla
         plp
         xce

IMVect   jmp   $123456
         rtl


;Tell assembler that we're done working in
;  8 bits.

         longa on
         longi on

;****************************************
;
IMSetup  entry
;
; Description: This little routine copies
;              system interrupt mgr vector
;              to JuMP instruction above.
;
;*****************************************


         lda   SystemIntMgr
         sta   IMVect+1
         lda   SystemIntMgr+1
         sta   IMVect+2
         rts
         end


;*****************************************
;
Data           data
;
; Description: This is data segment of this
;              demo program.
;
; The only interesting data is border color
; table & its index table.  The above code
; written so that it'd be easy to add more
;  border color changes.  The 'only' changes are
;  to set the scan line interrupt bit for the
;  additional lines, add the additional border
;  color to the border color table and add a new
;  index to the index table (IE: for 3 colors
;  set the color index table up to be 1,2,0).
;
;*********************************************
```

```
; RECT defining the box we draw on the screen

TheRect     dc
i2'(ScanLine1+1),0,(ScanLine2+1),320'
PenPatt     dc 32h'4444'  ;same blue as border


; The color table is the set of colors that the
;  screen border will cycle through

ColorTable dc i1'6,0'  ; Blue, Black


; Index table for which border color to display
;  with the next scan line interrupt.

NextColor  dc i1'1,0'

; Index of which border color to display next.

ColorIndex  dc i1'0'


; A temporary location for the border color
;  when we retrieve it from the battery backup
;  RAM.

OldBorderCl ds 2


; Location to save the system interrupt manager
;  vector.

SystemIntMgr ds 4


; The User IDs created for this demo when it
;  runs.

ProgramID  ds 2

PrivateID  ds 2

; Quit parameters

QuitParms dc i2'2'
          dc i4'0'
          dc i2'0'

         end
```

Letters



Losing Weight, Scuzziness, and Fame

Dear Ross,

Finally, a magazine with real Apple II hackers in mind.

After all the trouble with postage and orders, I finally received the disk and the magazine. However, I did notice that the number of pages has steadily decreased from July through September.

Now for a few comments on the September issue of 8/16. I congratulate Yvan Koenig for agreeing that there is a problem with the High Speed SCSI card. Good luck trying to convince Apple otherwise. Their favourite buzzword seems to be "arbitration". Perhaps *8/16* readers have comments on this. The delay can be anything up to 8 seconds long, and occurs every time you reboot. You can fix this by inserting a BUS_INIT call (INIT code 5) on the boot blocks, or in the ProDOS boot file. The init takes about 2 seconds, but at least you can live with it. (see my SCSI Part documentation for more details)

Plus, the START/STOP unit SCSI call is now restricted to CD only. So much for apparent ANSI X3.131-1986 compatibility, where START/STOP is an optional command for direct access devices (ie. all or none, not none or some). The only work around is to code a "generic SCSI command" which sends the START/STOP unit to the drive (START/STOP unit is how you park heads on hard disks without auto parking).

As a follow on from Yvan's next paragraph, I wrote a utility last year (called ILTS) which allows you to save your BRAM parms to the boot blocks. It is included on the same disk which includes my SCSI Part program, which I hear is currently doing the rounds over there. Also on that disk, is my resource manipulation utility, which was also written last year, AND my Pixie program (again 1989) which I use extensively instead of Nifty List (pre-3.0 version anyway, as I haven't fully checked out Dave's latest offering), as do many other Australian developers. PLUS, my FREDA debugging tool, which intercepts simple COP command calls for displaying text and register values while a program is running. And it's all FREEWARE!

I have included the disk with this letter in case you want to offer some of them on the next 8/16 disk.

It was quite frustrating trying to distribute these programs. As an example; I wrote SCSI Part in about June 1989. I then sent it to various people in Australia and the US, including A2-Central. It wasn't until a couple of weeks ago, that someone finally realised how handy it was, about six months after I'd given up plugging it. Amongst all the concern for the future of the IIgs, it's disappointing to find that the people in charge of software libraries, or distributing information about the IIgs, continually ignore what's on offer because they apparently don't have the knowledge to realise what's good and what aint! (Like long sentences? I do.) It seems unless your name is reasonably well known, no one takes you seriously. I suppose that's the advantage I have in Australia, where I'm fairly well known.

Anyway, keep up the good work.

Richard Bennett,
Sydney, Australia.

Dear Richard,

Apple II folks are getting a little paranoid, I'd say, though I guess it's understandable. It so happens that your first issue (July) was our thickest to date and your last (September), was our thinnest. We've been thinner than July both before and after.

The reason for this bouncing around has little to do with finances. I tell our authors the same thing I used to tell my writing classes when asked the age-old question, "How long?": "Long enough", I'd say, meaning long enough to adequately and clearly cover the material without verbosity. Thus our size is somewhat a function of the length of the articles in that issue.

September was shorter than I'd planned due to a canceled ad and a missed deadline (we didn't run VaporWare last month).

In a nutshell, the length of the September issue was more a function of article length and availability, not finances.

Thanks for your program submissions. They shall indeed be on our quarterly disk. In defense of the fine folks at A2-Central, they get thousands of programs and letters. That they can respond as well as they do amazes me. It is sometimes difficult to grasp the quality or significance of a submission based on a cover letter or even a quick perusal of the contents. That's why a certain level of "fame" does indeed add a little credibility.

I've not had time to look over your programs, but they really sound exciting. If it works out, I may feature them in our ToolSmith column.

Glad to have you aboard. Thanks for the letter and the goodies.

== Ross ==

Macro Madness, Praise, and Advice

Dear Ross,

I enjoyed your article "Magical Resources" in *8/16* Volume 1 Number 4. However, some of the macros used are a mystery to me. Particularly, the "deref" macro. Could you please print the contents of these macros in a future issue of *8/16?* I think that it would be important in future articles for the authors to give the macro definitions somewhere in the text if it is their own macro or to list where to find the macro definitions. I did find the ErrorDeath macro on an Apple II Tech Note.

I would also like a more detailed explanation on the PictCovert section of the "Magical Resources" program listing. In particular, what is this Linkin+0... Linkin+4, etc. The author did a very good job on all other aspects of the program, but I wish he could have explained the PictCovert section better.

One final point about *8/16* in particular that I think is very important. Joe Jaworski did a fine job telling the reader why the Resource Manager is important. I only wish other authors would do the same. I can't tell you how often I've tried to read an article about programming where the author launches into how to do it, but gives no introduction as to WHY you would want to. Looking over back issues of *8/16*, I can see that most of the articles do give good introductions, but if you look at the last issue of *CALL A.P.P.L.E.*, you can see a bad problem. There is an article called, "Roll Your Own Icons - Incorporating Icons into Your Applications". Apparently the title of the article is the only hint as to what the author will discuss. I've owned my IIgs since they first came out, but except for the Finder, I've never seen what I would consider separate icons anywhere. What does the author mean when he says that it might be a good idea to "roll your own icons?" How can I read an article that won't even explain what's going on?

All articles should start with what exactly is going on and offer reasons for doing these things. If the "Roll Your Own Icons" article tells how to put little faces or pictures in odd corners of my graphic screen while my program is runing, then it should have stated so at the beginning. But I have no patience with articles that won't get to the point.

I'm very glad to see that the editing in *8/16* is a good notch above that in *CALL A.P.P.L.E.* I'm looking forward to a long relationship with your fine magazine.

Sincerely,

Raymond Ross
Reno, NV

Dear Raymond,

Thank you very much for your kindest of comments, but it is not false humility for me to say that there is much about CALL A.P.P.L.E. that we aspire to and have not yet attained. It was really a remarkable publication.

Case in point: we blew it as far as the macros you mentioned. In fairness to my editorial staff, it is an easy thing to miss since deref, for example, ***is*** *a standard library macro in the Merlin assembler, but is not under Orca/M. I don't know about APW, but I think you see the potential for confusion.*

At any rate, here is the listing of some of the Picture Show macros (I purged tool call macros since I ***know*** *they are standard). I'm willing to bet that Joe has a special file of his own macros that he has in the AINCLUDES folder. This allows him to scan for standard macros in one fell swoop and grab any of his own, too.*

```
 MACRO
```

```
&lab ERRORDEATH &text
&lab bcc end&syscnt
 pha
 pea x&syscnt|-16
 pea x&syscnt
 ldx #$1503
 jsl $E10000
x&syscnt dc i1'end&syscnt-x&syscnt-1'
 dc c"&text"
 dc i1'13',i1'13'
 dc c'Error is $'
end&syscnt anop
 MEND

 MACRO
&lab link &SizeInput,&SizeLocals
&lab ANOP
 AIF C:&SizeLocals,.a
 LCLC &SizeLocals
&SizeLocals SETC 0
.a
linkin equ &SizeLocals+3
linkout equ linkin+&SizeInput
 tsc
 sec
 sbc #&SizeLocals
 tcs
 phd
 inc a
 tcd
 MEND

 MACRO
&lab unlink
&lab ANOP
 pld
 tay
 lda linkin-1,s
 sta linkout-1,s
 lda linkin-2,s
 sta linkout-2,s
 tsc
 clc
 adc #linkout-3
 tcs
 tya
 cmp 1
 MEND

 MACRO
&lab wordspace
&lab ANOP
 pea 0
 MEND

 MACRO
&lab longspace
&lab ANOP
 pea 0
 pea 0
 MEND
 MACRO
&lab copylong &addr
&lab ANOP
 lda 1,s
 sta &addr
 lda 3,s
 sta &addr+2
 MEND

 MACRO
&lab deref &addr
&lab ANOP
 lda &addr
 sta <4
 lda &addr+2
 sta <6
 lda [<4]
 sta <0
 ldy #2
 lda [<4],y
 sta <2
 MEND
```

As you can see, the deref macro "dereferences" a handle, meaning that it returns the actual memory address the handle points to.

The references to Linkin+0... to Linkin+4 have to do with the "link" and "unlink" macros. These little buggers create and tear down a structure called a "stack frame".

A stack frame is simply a mechanism whereby parameters and variables local to a function, subroutine, subprogram, or procedure may be stored on the stack and then discarded when finished. To access one of the parameters or variables within the stack frame, you look a certain distance from the stack pointer.

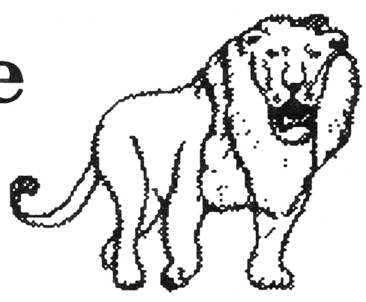
That is where Linkin+0, etc. comes from. On page 16 Joe defines what variables live where, that is where they reside on the stack relative to the stack pointer when the "link" is established. Thus they are EQUated to offsets from Linkin.

NOTE: There is a subtle yet serious bug in Joe's code. Please check out the Insecticide notice for an explanation.

As for your advice about article introductions - I agree wholeheartedly. The WHY is just as important as the HOW. Letters such as yours help us keep these fundamentals in mind.

Thanks! == Ross ==

KAT will sell no drive before it's time...



KAT will not ship a hard drive without first:

• Conferring with you about your entire system and setting the drive's interleave so as to insure optimal preformance *for you.*
• Discussing the various partioning options and then *setting them up to fit your specifications.*
• Depositing 20 megabytes of freeware, shareware, the latest system software, and all sorts of bonus goodies on the drive.
• Testing the drive for 24 hours before shipping it out.

KAT drives come in industrial-quality cases that have 60 watt power supplies (115-230 volts), cooling fans, two 50 pin connectors and room for another half-height drive or tape back-up unit. We also include a 6 ft. SCSI cable to attach to your SCSI card. You get all of this plus a one-year warranty on parts and labor!

SB 48 Seagate 48 meg 40ms	$549.99
SB 85 Seagate 85 meg 28ms	$698.99
SB 105 Quantum 105 meg 12 ms	$849.99

Looking for an even *hotter* system? Call and ask for a quote on our 170, 300, & 600 megabyte Quantum drives!

So ya wanna build yer own? Let KAT provide you with the finest parts available...

SB Case 2 HH Drives 7w 5h 16d	$139.99
ZF Case 1 HH Drive 10w 3h 12d	$169.99
48 meg HD Seagate 40 ms 3.5" SCSI	$349.99
85 meg HD Seagate 28 ms 5.25" SCSI	$469.99
105 meg HD Quantum 12 ms 3.5" SCSI	$669.99
T-60 Tape Teac 60 meg SCSI	$449.99
with hard drive	$424.99
3.5" to 5.25" Frame	$ 12.50
Cable 25 pin to 50 pin 6 ft.	$ 19.99
50 pin to 50 pin 6 ft.	$ 19.99

Programmers! Check our prices on your favorite development packages and accessories...

Byte Works

Orca C	$89.99
Orca M	$44.99
Orca Pascal	$89.99
Orca Disassembler	$34.99

Roger Wagner Publishing

Hyperstudio	$94.99
Macromate	$37.99

Stone Edge Technologies

DB Master Pro $219.99

Other software and accessories:

Vitesse, Inc.
Excorciser, virus detection system $ 29.95
Renaissance, hard disk optimizer $ 34.95
Guardian, program selector and disk utilities $ 34.95

Applied Eng. Transwarp GS $289.99
Keytronic 105 Key ADB Keybrd $139.99

Quickie, terrific hand scanner (400 dpi, 16 grays) $249.99

Computer Peripherals
ViVa24, 2400 baud, 100% Hayes compatible modem (comes with a FIVE YEAR Warranty) $139.99

1 meg SIMMs 80 ns $89.99
1 meg X 1 80 ns 8/$79.99

Call the KAT at (913) 642-4611 or write: KAT, 8423 W 89th St, Overland Park, KS 66212-3039

VaporWare

NOTE: This column is primarily for entertainment purposes and statements expressed within may be based on rumor or innuendo. Furthermore, the views of the author do not necessarily reflect the views of *8/16* and/or its editorial staff.

by Murphy Sewall

From the September 1990 APPLE PULP

K-12 Macs.

The first two models in Apple's long awaited line of low cost Macs are set for an October 15 unveiling. The monochrome "Mac Classic" is a close cousin of the popular SE with an 8 MHz 68000 CPU and a 3.5 inch 1.44 Mbyte SuperDrive. Apple has not yet decided whether to include one or two Mbytes of RAM for the $1,295 list price. The more powerful color machine, codenamed "Pinball," will be a modular design closely resembling the Apple IIgs. Priced between $3,000 and $4,000 with a SuperDrive and 40 Mbyte hard disk, the Pinball is based on a 20 MHz 68030. After the first of the year, Apple will introduce a $2,750 20 MHz 68020 system currently known as the Mac LC. All three K-12 machines have a single expansion slot which, in many cases, will be filled with the optional Apple II card which Apple has been field testing for about two years. In a related move, Apple plans to replace the IIcx, possibly with a design that retains the current system's features but can be manufactured and sold for a significantly lower price. *- PC Week 6 August and InfoWorld 13 August*

Mac in a II.

Before Apple offers it's IIgs card for the Macintosh (after the first of the year), Cirtech promises to offer a Macintosh card for the IIgs (in December). Cirtech's Duet uses a 68020 processor, a custom ROM, and up to 8 Mbytes of RAM. Duet recognizes standard Apple peripherals using the IIgs's 65816 for I/O processing. A socket is available for an optional 68882 math coprocessor. Cirtech claims the system will outperform a Mac IIcx. Price information is not yet available. *- A2-Central August*

Apple II Marketing Strategy.

Late August is said to have been the prospective date for a decision on Apple's new marketing strategy. Advocates for returning to the company's roots (computing for fun as well as for profit) are being listened to seriously by the firm's most senior management. Apple is developing and testing new CPUs (plural), but the decision of when and what to market depends on more than just technological issues. The unreleased IIgs operating System Disk (5.03) in use by Apple's support group at the recent KansasFest developers conference, appears to contain most of the new features touted in early System Disk 6.0 rumors. Could it be that Apple has just decided to change the designation for the next release? In addition to improved memory utilization (memory is no longer fragmented on bootup), direct access to the modem port, and a few bug fixes, some interesting new tools also are included. *- A2-Central and notes found in my electronic mailbox*

500 Word Per Minute Typing.

Caere Corporation will soon begin shipping the "Typist," a hand-held scanner bundled with character-recognition software that uses keyboard interrupts to direct characters directly into any application. The 20K RAM resident (desk accessory on the Mac, terminate and stay on the PC) Typist has a 300 dot-per-inch five inch scan head and a virtually transparent interface. The Mac version ($695) is slated to ship in September and the PC version ($595) will be available in the fourth quarter. *- InfoWorld 6 August*

Insecticide

Thanks to the ever-vigilant Nate Trost for pointing out the subtle bug in Joe Jaworski's "Magical" Resources" article in the June 1990 issue. If you'll get out your red pens and turn to page 15: Joe's code pushed the resource ID followed by the type. This is incorrect, but happened to work in his particular program. His comments **were** correct, but his labels were mixed up.

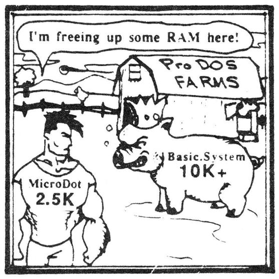
As page 45-56 of the *Apple IIgs Toolbox Reference, Volume 3* points out, the correct order for the code is really:

```
      pea    #0
      pea    #0     ;long word result space
* word length resource type first
      pushword resType
* then long word resID you've assigned
      pushlong resID
      _LoadResource
```

And don't forget, of course, that LoadResource returns a handle to the resource. Be sure and check for a nil handle, too, in case there was a problem.

MicroDot just $ 29.95 plus $2.50 S&H

The Logical Replacement for BASIC.SYSTEM

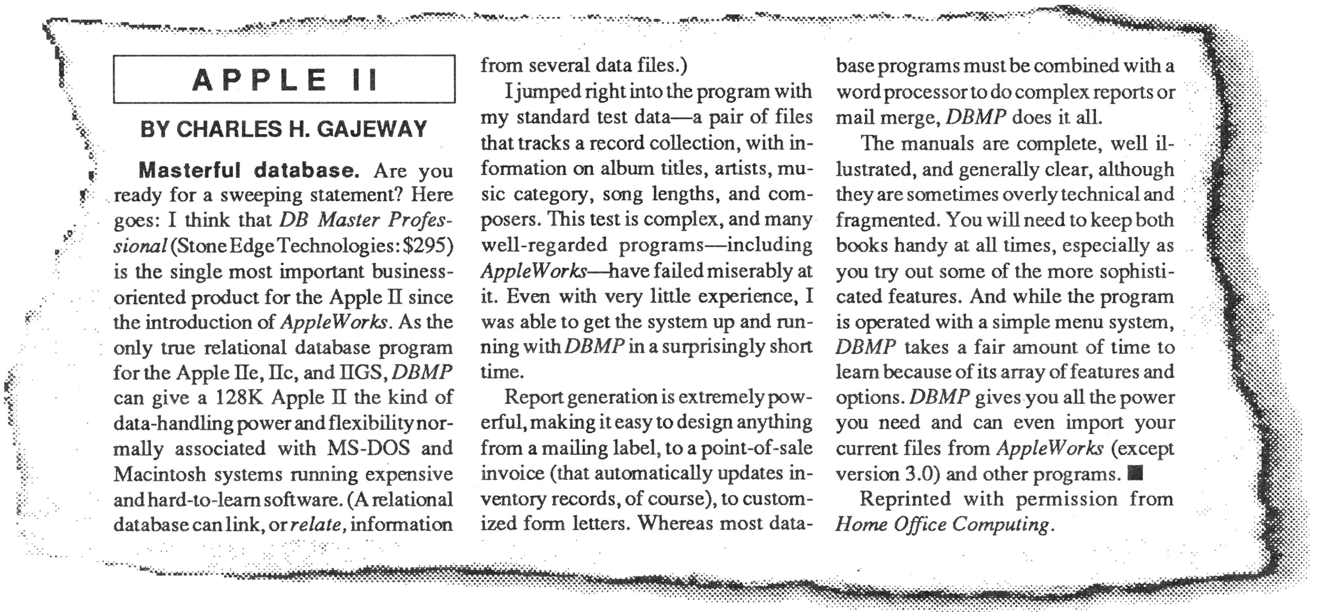


Just **2.5K** in size, but more powerful than BASIC.SYSTEM. Imagine doing BASIC overlays simply by specifying the file name and the line number where you want to overlay. How about loading an array of directory names at machine language speed. You get this and total control over ProDOS that is impossible with BASIC.SYSTEM. Works with Program Writer ($42.45. Both for $59.95 + S&H). Love it or get your money back! Inexpensive publishers' licenses.

Free Catalog and Details Dealer Inquiries Invited

Kitchen Sink Software, Inc
903 Knebworth Ct. Dept. 8
Westerville, OH 43081
(614) 891-2111

"...the single most important business-oriented product for the Apple II since *AppleWorks.*"



APPLE II

BY CHARLES H. GAJEWAY

Masterful database. Are you ready for a sweeping statement? Here goes: I think that *DB Master Professional* (Stone Edge Technologies: $295) is the single most important business-oriented product for the Apple II since the introduction of *AppleWorks*. As the only true relational database program for the Apple IIe, IIc, and IIGS, *DBMP* can give a 128K Apple II the kind of data-handling power and flexibility normally associated with MS-DOS and Macintosh systems running expensive and hard-to-learn software. (A relational database can link, or *relate*, information from several data files.)

I jumped right into the program with my standard test data—a pair of files that tracks a record collection, with information on album titles, artists, music category, song lengths, and composers. This test is complex, and many well-regarded programs—including *AppleWorks*—have failed miserably at it. Even with very little experience, I was able to get the system up and running with *DBMP* in a surprisingly short time.

Report generation is extremely powerful, making it easy to design anything from a mailing label, to a point-of-sale invoice (that automatically updates inventory records, of course), to customized form letters. Whereas most database programs must be combined with a word processor to do complex reports or mail merge, *DBMP* does it all.

The manuals are complete, well illustrated, and generally clear, although they are sometimes overly technical and fragmented. You will need to keep both books handy at all times, especially as you try out some of the more sophisticated features. And while the program is operated with a simple menu system, *DBMP* takes a fair amount of time to learn because of its array of features and options. *DBMP* gives you all the power you need and can even import your current files from *AppleWorks* (except version 3.0) and other programs. ■

Reprinted with permission from *Home Office Computing*.

DB Master *Professional*

Stone Edge Technologies, Inc.
P.O. Box 3200 • Maple Glen, PA 19002 • (215) 641-1825

GENESYS NEW! **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.

From the House of Ariel

• 8/16 on Disk •

We don't have the room to even come close to telling you what goes into the disk every single month. We estimate that by the end of our first year we'll have delivered approximately 8 megabytes of source code, utilities, articles, and other goodies for Apple II programmers. That works out to less than $9 per megabyte. *I* think it is the deal of the century, but since I'm naturally quite biased, I thought I'd tell show you the kind of feedback we're getting about it...

"I have found it to be a fantastic investment: I've never had soooo much information in one place before..." - Michael W. Faulkner, Berlin, Germany

"You guys are simply outdoing yourselves..." - Robert Todoroff, St. Louis, MO

"I can't live without it!" - Robert Santos, Miami, FL

The magazine you are now holding in your hands is but a small 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.

Highlights from the last four disks (so far every disk has had more than 600K of material!):

• **Sept '90:** 8 bit - Jerry Kindall's Generic Startup routines and the complete source code to Karl Bunker's DOGPAW
16 bit - Jason Coleman's shareware resource editor, LLRE; Morgan Davis's universal shell routines.

• **Aug '90:** 8 bit - Jerry Kindall's Generic Shutdown routines for assembly (this is GREAT); a complete, working Forth language compiler (Uniforth); Ross's FN Local and FN SetEOF for ZBasic programmers (A classic... hehehe - guess who's writing this!)

16 bit - Doni Grande's extended keyboard code; Jay Jennings' extended control routines; and - believe it or not - **Nifty List v. 3.0, by Dave Lyons.**

• **July '90:** 8 bit - the assembly source to Super Selector, which includes code to eject 3.5" disks; the ZBasic code for DrawPoly.FN, a super neat, flexible DHR and hires poly plotter; the demo to Shem the Penman's Guide to Interactive Fiction

16-bit - an updated Orca/APW shell command, COPY; Console Driver demo (with source and an information file (this is neat!); Steven Lepisto's Illusions of Motion Number Three.

• **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. Non-North American orders add $15 for 1 year, 8$ for 6 months, and $5 for three months. All disks are shipped first class.

• *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 *The 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" or 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.

• *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.

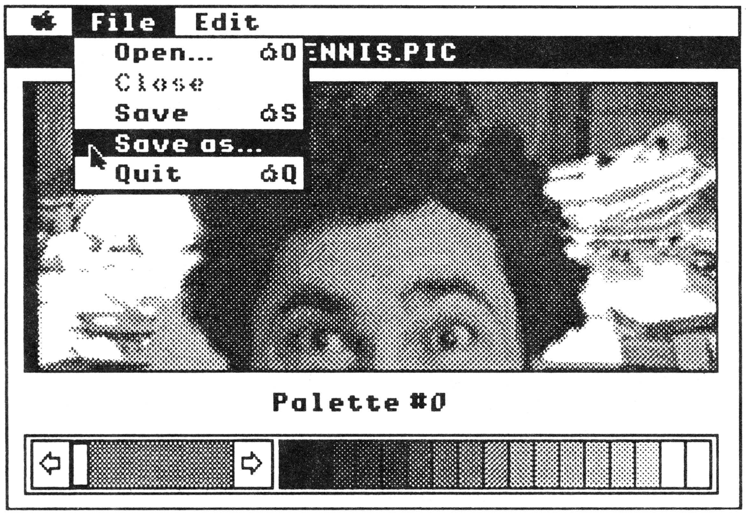
All back issues are $3.00 each (postage and handling included except for non-North American orders. Those of you on other shores please add $1.50 extra per issue).

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.

Ordering Info:

To order, just write to: **Ariel Publishing, Box 398, Pateros, WA 98846** or call **(509) 923-2249.** Our fax number is **(509) 689-3136**.

We accept Visa, MC, personal checks, IOU's, institutional purchase orders (for those of you in institutions), RAM chips, TransWarp GS's, Apaloosa's, hats from around the world, programming work, etc. Be creative if you're broke.



Applesoft™ Never Looked So Good!

The Call Box TPS™ (*Toolbox Programming System*) gives you the tools to look and sound your best. Make your own Applesoft BASIC desktop applications which look and sound like professional programs.

Over 1000 toolbox calls have been added to Applesoft BASIC which gives you, the BASIC programmer instant access to the Apple IIgs toolbox in a simple and flexible way. You can use the Memory Manager, Miscellaneous Tools, Tool Locator, Quickdraw II, Desk Manager, Event Manager, Scheduler, Sound Manager, Desktop Bus, Text Tools, Window Manager, Menu Manager, Control Manager, Quickdraw II (aux.), Line Edit, Dialog Manager, Scrap Manager, Note Synthesizer, Note Sequencer, A.C.E., Standard File and much more. In addition to all the tool calls you have access to ProDOS 16 and GS/OS commands at the same time that you have access to ProDOS 8 commands. You can even load and run relocatable shell applications from within the Call Box BASIC environment.

The Call Box TPS includes the BASIC interface, WYSIWYG Window, Dialog, Menu and Image editors, Disk and system utilities plus demos and tutorials. The Call Box TPS comes on 3 - 3.5"disks with a 140+ page hard cover ring binder manual. Requires 1 megabyte min. and GS/OS V5.0.2 min. Call Box is supported by a programmers association which provides its members with disks and documentation designed to educate as well as illuminate.

The Call Box TPS $99.00

So What Software™

10221 Slater Ave. Suite 103 Fountain Valley, CA. 92708
(714) 964-4298 VISA/Mastercard accepted

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 t0 port 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

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

Ariel Publishing	40,41
Bringardner Data Products	5
Direct Micro	44
Kitchen Sink Software	38
Ron Lichty	5
LRO Computer Sales	24
Night Owl Software	2
So What Software	42
SSSi	39
Stone Edge Software	38

BULK RATE
U.S. POSTAGE
PAID
PATEROS, WA
PERMIT NO. 7

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

Laser 128 is a registered trademark of Video Technology Computers, Inc. Apple, Apple IIe, Apple IIc and Imagewriter are registered trademarks of Apple Computer, Inc.

S