

September,
1990
Volume 1,
No. 7

8 / 16

The Journal of Apple II Programming

\$4.00

The Kansas Report:

Uncle DOS looks funny with wet hair

all	The Publisher's Pen, by Ross W. Lambert	3
	<i>re: the Kansas Fest Report: What's New for the II</i>	
GS	The ToolSmith, by Ross W. Lambert	7
	<i>re: Nifty List v 3.0 and LLRE reviewed and explicated</i>	
8 bit	The ZBasic Zealot, by Ross W. Lambert	12
	<i>re: FN Local and FN SetEOF</i>	
GS	To Shell With It, by Morgan Davis	17
	<i>re: creating universal shell utilities</i>	
8 bit	Generic Shutdown, by Jerry Kindall	24
	<i>re: a generic shutdown routine for 8 bit assembly</i>	
all	Letters to the Editor	27
8 bit	Applesoft Auto Wordwrap, by Jerry Kindall	30
	<i>re: automatic wordwrap for Applesoft text output</i>	
all	Hired Guns	38

Ariel

Publishing

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

New kit restores your Apple IIGs

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

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

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

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

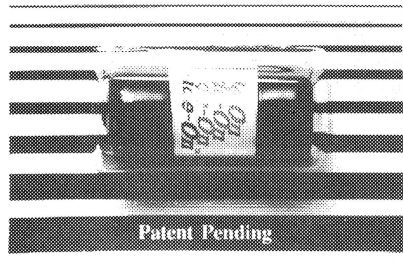
Typically, our customers have reported that the original equipment batteries have an average life expectancy of 2 to 3 years. This is about half as long as they were supposed to last. Slide-On replacement kits include Heavy Duty batteries which should provide for a longer battery service life.

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

Nite Owl's
**Slide-On
Slide-On
Slide-On
Slide-On**
Brand

**Battery Replacement Kit
for
Apple IIGS Computer**

- Fantastic Savings
- Easy Installation
- No Solder Required
- Complete Instructions
- 10 Year Shelf Life
- Top Quality Lithium



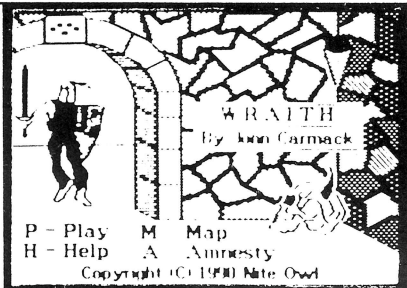
Patent Pending

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

New

WRAITH
Adventure Game

Special
Introductory
Price \$9.95*

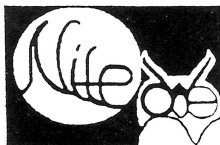


This graphic adventure game comes complete on a single 3.5 inch disk with on-screen instructions, a map, demo play option, and dungeons which were too vast and expansive to fit on 5.25" disks.

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

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

Please give us a call today at: (913) 362-9898
Photo-Copyable FAX: (913) 362-5798



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

(Cut & Paste Address Label)

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

School Purchase Orders are Welcome.

Ship to:

Telephone #: _____

Credit Card or PO#

• Bill To •

Cash, Check,
Money Order

VISA

Master Card

Purchase
Order

Expiration Date

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

Prices may Change without notice.

8/16

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

Publisher & Editor-in-Chief
Apple IIgs Editor
Classic Apple Editor
Contributing Editors

Ross W. Lambert
Eric Mueller
Jerry Kindall
Jay Jennings
David Gauger
Steve Stephenson
Mike Westerfield
Cecil Fretwell
Tamara Lambert
Karen Redfield

Subscription Services

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.

WARRANTY and LIMITATION of LIABILITY

Ariel Publishing, Inc. warrants that the information in 8/16 is correct and useful to somebody somewhere. Any subscriber may ask for a full refund of their last subscription payment at any time. Ariel Publishing's LIABILITY FOR ERRORS AND OMISSIONS IS LIMITED TO THIS PUBLICATION'S PURCHASE PRICE. In no case shall Ariel Publishing, Inc. Ross W. Lambert, the editorial staff, or article authors be liable for any incidental or consequential damages, nor for ANY damages in excess of the fees paid by a subscriber.

Subscribers are free to use program source code printed herein in their own compiled, stand-alone applications with no licensing application or fees required. Ariel Publishing prohibits the distribution of source code printed in our pages without our prior permission.

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

Apple, Apple II, IIgs, IIc, IIc+, IIe, AppleTalk, and Macintosh are all registered trademarks of Apple Computers, Inc.

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

The Publisher's Pen

by Ross W. Lambert



I am writing this column the day after my return from KansasFest (and immediately following the first full night's sleep I've had in four days). Egads. What a conference it was.

Meeting so many people was a total gas, but it also really impressed upon me the weight of responsibility the house of Ariel carries as journalists.

That's not to say that I'm going soft on you. On the contrary; my English and journalism background almost makes it impossible for me to deny the "Fourth Estate's" responsibility to ferret out the truth in matters great and small. But with that constitutional right comes a great burden - the burden of proof and factuality.

The event that precipitated all of this will seem a trifle to most of you; an error in Murphy Sewall's VaporWare column of July. He reported that all of the Mac programmers at Beagle Bros, Inc. had departed from their employ.

A knowledgeable source informed me that this was patently untrue. Now don't get me wrong; Murphy is most definitely entitled to his opinion, as are we all. But we did not include any header or disclaimer to Murph's column to indicate that it was substantially speculation based on **rumors**.

For that, I'd like to apologize to Mark Simonsen and the folks at Beagle.

I want 8/16 to be a trustworthy source of information. Part of the charm of a column like Murph's is the sometimes outrageous expression of opinion. This is okay, but it should be obvious as such. I'll make sure it is

from now on. I plan to put a bold disclaimer at the top identifying that "VaporWare" is what it is. Murphy is a perceptive observer of the industry, but his observations in the column are based on rumors or reports of rumors. Just because *PC World* prints something doesn't mean it is true. So take VaporWare for what it is - an entertaining, somewhat askance view of the industry not to be taken too seriously.

Furthermore, and perhaps most importantly, you all are free to express your opinion's in letters to the editor, too.

Hello? Is anybody home?

Which leads me to a second point. Letters. We want 'em. We'll even handle your technical questions - that's why I arranged for Cecil Fretwell to take Mike Rochip's place as our resident guru. And we want you to express yourselves, too. For example, if you disagree with our opinions then do so - in print. We'll give you the space. That is important to the health of any community and helps check the spread of misinformation.

Now back to the conference... Keep in mind as you read this that I can only reliably report on the sessions I attended. I wish I could've gone through 'em all. Furthermore, keep in mind that the absolutely positively best part of the entire conference for me was to meet and mingle with my heroes. I got to play Roger Wagner's guitar and listen to a 1:00 AM HyperStudio demo (Roger, you amaze me). I got to go see Arachnophobia and sit two chairs away from Randy Brandt (who wanted to come back with a plastic spider on a string). And I got to listen to Eric Soldan play some Bach on a slightly out of tune piano. In spite of the piano, it was totally astounding. Eric is multitalented to-da-max.

I attended the Iigs College on Thursday. It was very good, although if they ever have another organized the same way, I'd want to suggest the following two things to y'all: If you are a rank beginner at Iigs and/or desktop programming, then spend a little time boning up ahead of time. For the true neophyte, the beginner's track was a little too fast paced. For anyone with almost any desktop programming experience, however, you ought to take the advanced track. For this open-ended and less structured set of sessions, it is important to bring a boat load of questions. It's a chance to let someone else save you time by helping you with your sticky problems.

The Apple DTS crew did a great job (C.K. Haun gets my vote as best overall presenter), and they even gave us free goodies: a "Moof!" mousepad with a dogcow on it, a complete set of Apple II Tech Notes, and a KCFest disk with a zillion tools on it including GS Bug. These were worth the price of admission, lemme tell ya.

One dynamite new product demo'd at the College and elsewhere was Dave Lyons Nifty List version 3.0. Dave works for DTS and just couldn't help using the software (along with everyone else - he didn't have to try to sell it, it sold itself). It is immensely useful and is one of the few items I'd mark as indispensable for the Iigs programmer.

I was impressed enough to include it in the first edition of our ToolSmith column. Look for it elsewhere this issue.

The general sessions began on Friday, and included several by Apple that I cannot talk about at all; we had to sign non-disclosure agreements. They were exciting, and I will say this much: Iigs sales are going to pick up a lot. If Apple's marketing department does even half as well at pushing the new goodies as I want them to, there will most definitely be a resurgence of interest in the II line.

And the rumor mill is actively grinding out more juicy tidbits every day. I'll leave those to Murph, however.

My first session was entitled "Apple Iigs System Software Update". This appears to be a regular item on the agenda at each conference. I think it is a little misnamed; there was no updating done, really. The Apple folks merely overviewed the components of the latest official release. There is nothing wrong with this, but a few attendees went in expecting to hear about unreleased or future versions of system software. Although Microsoft seems to prosper by semi-officially leaking information, I guess we'd all better get used to the fact Apple just does not do that. I for one have given up trying to play the "tell me a secret" game.

This little hacker went to market...

The next session I went to was my own: Marketing for Small Developers. I'd like to publicly blast Tom Weishaar for not giving me two hours (just kidding, Tom, though I wish you had). I really only covered the basics and ran out of time to get into the more treacherous

waters. There appears to be sufficient interest in the subject that I have decided to convert my seminar to article form and serialize it within this column. I'll try to keep it all as practical and useful as possible.

At 11:00 AM I stumbled into Jim Mensch's animation tools session. This man is a wizard, even if he is a "scum sucking elitist pig" by his own admission. If you want to know the truth, I was more excited about his new toolkit than anything else at the whole conference. Jim's "AnimateGS" is going to be a simple mechanism whereby we can do high quality animation with minimal programming overhead. It is a really hip idea, I think, because it will allow generalists like me to add credible animation to their 'wares without having to spend six months learning the tricks of the masters. We'll have to wait a while (it is not even to the alpha stage yet), but even the demo was really exciting.

With the exception of the lunchtime speech by Jane Lee, the entire rest of the day (as far as you are concerned) was signed away into non-disclosed oblivion. C'est le vie.

Jane Lee, however, was charming and encouraging. She is the official Apple II Marketing Geek. (I've said that of myself, Tom Weishaar, Rajiv Mehta, and now Jane all with tongue in cheek. It's *not* an insult, though it may be getting old. Time to move on to another colorful description.) Jane reported on her progress towards moving the Apple II forward in the corporate consciousness. It is happening, though I visualize the process to be somewhat like turning over an elephant. Jane has a rope around the neck pulling with all her might along with the help of Ralph Russo, the new Apple II Overseer and Grand Poobah. John Sculley is on the back side pushing with a couple of fingers. Many parts of the elephant are moving, many parts are not. This is to be expected.

A well placed kick...

After talking with several developers, it is clear that one part of the corporate elephant that is not yet moving in the right direction is Evangelism. I have an anonymous but highly

placed and reliable source in the industry who told me that their evangelist is still counseling educational software firms to go to the Mac. One big-time educational software firm has dropped all further Apple II development because of it.

The company involved must really have some warped perceptions of schools. I can see why Apple would want to encourage Mac educational software development, but they must be very careful to encourage **parallel** development. If a company is producing Mac educational software, it is almost ludicrous not to do an Apple IIgs version at the same time. It opens up a broader market with little, if any, extra development time.

It is entirely possible (maybe even probable) that the company involved has misapplied the evangelist's advice.

"Jane Lee reported on her progress toward moving the Apple II forward in the corporate consciousness. It is happening, though I visualize the process to be somewhat like turning over an elephant."

But the point remains that the Apple employee in question left himself open to that kind of terribly erroneous misinterpretation. The time has come for evangelism to evangelize software firms to produce GS software. The situation right now is quite similar to the Mac's circa 1986.

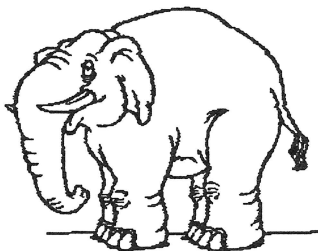
We've got a great machine with some really great features, but the developers need some encouragement. I think Apple needs a IIgs evangelist (or two).

I have never hidden the fact that I hack the Mac and Ariel Publishing, Inc. even has a Macintosh publication. Those facts alone, I think, should lend some credence to what I am saying. I am not a wild-eyed radical wishing for a return to 1982. I am a businessman and an opportunist. There are powerful, logical, bottom-line oriented arguments for developing Apple IIgs software right now. Evangelism is not fulfilling their mission if they are not in tune with that.

Keep pulling, Jane.

There were several good seminars on Friday night, including Vidar Jorgensen's "Extending the Life of the Apple II", an AppleTalk session with Brian Fitzgerald, and a sound and animation Q & A session with Chris McKinsey, Lane Roath, and Bill Heineman which then turned into A2-Centralite Jay Jennings's long

anticipated "All Night HackFest". Fortunately for all concerned, the doors had to be locked some time around 2:00 AM. Most folks were exhausted beyond all belief, anyway, and I didn't last anywhere near that long. If you must know the truth, I couldn't attend any of these as I was "unavoidably indisposed".



Saturday

I felt better on Saturday, though, and made it to the SynthLab session, Llew Roberts' CD-ROM session, and the Apple IIGs media integration session.

SynthLab is beyond description. It is a MIDI sequencer, a 16 channel mixer, a synthesizer controller, and mucho more. I'm sure I don't even understand the half of what it does. Apple music guru Mark Cecys has done a wonderful job putting together a package that will put the IIGs on the map in the music world. Although the product is not shipping yet, it was mucho impressivo nonetheless.

The media integration and CD-ROM sessions were not technically enlightening, but were really motivational. I've always loved multimedia projects since I first put one together in graduate school. These sessions reminded me of their absolutely captivating power. We got to preview a Houghton-Mifflin project that is truly astounding.

The only thing that worries me is that "media integration" (the newest buzzword) appears to be falling into the domain of the mega-houses (i.e. those that can afford to hire a camera team and buy expensive hardware like laser disk players, etc.). There are relatively inexpensive alternatives (licensable video and sound libraries and VCR controllers, for example), and I think if Apple really wants media integration to proliferate, they might want to consider putting

such libraries together so that we small to medium sized developers can get access to them without losing our shirts. It would be pricey for Apple, but they'd get a good portion of the cost back in license fees. It's a good deal for us because a license doesn't cost nearly as much as a full scale video production team.

As you might expect, there was "much, much more". I'm sure I forgot something important and significant, too. But I'll be sure and relate them to you as I remember them (and after I get just a little more sleep).

== Ross ==

P.S. Okay. I've had a little more sleep now. I forgot to mention that there was a horrendous rain storm before the Royals game. Didn't matter much to me because Bo Jackson was hurt, anyway. Guess he did it one too many times or something. We therefore crawled into buses and cars and boogied off to the movies. That's where we saw *Arachnophobia*. If you have the slightest fear of spiders, don't go see this flick. Otherwise it is a hoot.

Rewarding the Faithful

By the way, please allow me a moment to give a blatant plug for a faithful and consistent advertiser, i.e. Night Owl Production's Bob Shofstall. He has put together two disks that my brother-in-law has fallen in love with - *Wraith*, an adventure game that costs like \$9.95 or some ungodly low amount, and his latest release, *The Nite Owl Journal*. This latter disk is an eclectic 680K of goodies ranging from another adventure game to Applesoft programming utilities to a mailing label program. It is a *lot* of material for the low, low price of \$9.95. Call Bob and say the following words slowly: "Long live 8/16 and their advertisers. Send me the *Night Owl Journal*."

Bob also has replacement batteries for your GS (I've got mine, you got yours?) and several other goodies.

By the way, my bro-in-law is an adventure game addict and is one of the better players I've ever seen. He has over 25 hours into *Wraith* and calls it "top of the line".

Bob's ad is on the inside front cover if you need more details. Here's the most important detail of all, though:

Nite Owl Productions (913) 362-9898



The ToolSmith

Mega Power for Mini Bucks

by Ross W. Lambert

Due to popular demand, we are inaugurating this semi-regular column, henceforth dubbed *The ToolSmith*. The purpose here is to survey the software development landscape and not only review the environments and utilities available, but also dig into some of their more esoteric and powerful features. I hope to con (er- make that "convince") some of the developers of these packages to reveal their innermost secrets. Though we'll certainly be doing outright reviews from time to time, we'll also strive to make this a "how to" kind of series.

In a nutshell then, our goals here are to help you ferret out what you really do need to buy and then to help you get the most of it when you do.

So let's dig right in.

More than a trip to the movies...

One of the best side benefits of the A2-Central developers conference was being able to see professional programmers using and demonstrating their favorite tools. This was more helpful than you might imagine. I was skeptical and downright fearful of several of the hot new products until I saw them up close.

And lest you fear that I'm going to recommend you mortgage your house to get them all, let me point out ahead of time that two of my favorites are inexpensive shareware offerings (Nifty List and Low Level Resource Editor). The third is a \$30 APDA product (GSBug). We'll look at the shareware this month and GSBug next month. If you bought the whole ball of wax it would still be a paltry \$70. The time you save will be more than worth it.

Nifty List Niftier

Most GS programmers have at least heard of Nifty List by Apple, Inc.'s David Lyons. I used to think of it as simply a "glorified monitor" for the IIGs. Boy, was I an ignorant slimeball.

Allow me to digress a moment and encourage 8 bit programmers to continue reading - Nifty List is darn useful for 8 bit folks using the GS as their development platform.

Getting Resourceful

As soon as I started working with resources on the IIGs I soon realized how important Nifty List can be. But I'm getting ahead of myself.

Nifty List has the reputation of being hard to learn because it is so powerful. If that is what is holding you back, you're missing out on a lot of programming help due to groundless fears.

You can make Nifty List as easy or as hard as you want it to be (and that's the way the best programs should be, really). The fun thing for me is that it became *invaluable* just moments after I put it on my hard drive. I did not then nor do I now know the deep dark secrets surrounding its more mystical uses. I may never - but I'm still putting it to good use in the mean time.

Installation

Nifty List is a CDA that you just tuck into your /System/Desk.Accs folder. Installation is therefore a piece of cake. Version 3.0 comes with two optional "module" files (I'll explain what those are in a little while) that need only be tucked into the same folder.

Use and Abuse

Once Nifty List is installed and ready (did you re-boot? Or do you have InitRunner?) now all you do is just code as normal - until such time as you want to test your program/DA/init/ whatever. Once you launch it, you can do the three fingered salute (OPEN-APPLE/ CONTROL /ESC) and jump to the CDA menu. Select Nifty List, and you'll see the NL> prompt along with author Dave Lyons' title screen.

Getting Down and Dirty

Rather than tell you all about the plethora of commands that are available to you at this point, let's look at how Nifty List has already helped me.

It may seem a silly bug (is there such a thing as an intelligent bug?), but I was having trouble finding a cursor resource attached to my latest work-in-progress.

`_LoadResource` was returning an error every cotton-picking time.

Enter Nifty List.

As soon as I launched the app, I went into Nifty List and typed "Oi" (that's a zero and a lower case i). This command returns information about every handle allocated in memory. I figured that this would at least reassure me that my resource was indeed there. (A programming digression here: the resource didn't necessarily have to be there. Depending on the state of memory, the resource's flags, etc., the Resource Manager might not actually load the thing until the very moment you do a `_LoadResource`. In this respect the Resource Manager provides a limited form of virtual memory.)

As Nifty List dutifully pointed out, my resource was sitting contently in memory minding its own business. Figure 1 is a screen dump of what

Nifty List revealed. The Oi command (henceforth dubbed the "oink" command) lists all of the handles in memory, their address, their size, flags, their owner's ID, and the owner's path. The "i" part of the command stands for information. "Oi" says to Nifty List, "Give me info on everybody". 5000i would be saying, "Give me info on desk accessories only, please". You don't need to memorize that because David spelled it all out in his nice documentation file.

Best of all for those of us munging around with resources, Nifty List tells you the resource type by number and name as well as the ID assigned to that particular resource.

Perhaps I'm just a paranoid programmer, but this alone was worth the price of admission.

Another nicety is the ability to auto-dereference a handle. Take my cursor demo, for example (please!). If I actually wanted to find my cursor data in memory, all I have to type is the cursor's handle followed by a caret (^), a colon, and an h.

Like this: `E069C4^;h`

Figure 1: The Nifty List Oink Command (Oi)

```
NL> Oi
handle addr  size  flgs ownr path
E11700 000000 000800 C000 0000 (memory manage
E06CBC 002400 000800 C001 520A NIFTYLIST.CDA
E1196C 009A00 002600 C013 3201 (GS/OS)
E0676C 0108AA 0002D1 4010 1003 RCURSOR.DEMO
E06CBC 002400 000800 C001 520A NIFTYLIST.CDA
E1196C 009A00 002600 C013 3201 (GS/OS)
E0676C 0108AA 0002D1 4010 1003 RCURSOR.DEMO
...
E069C4 011006 000096 4000 1003 RCURSOR.DEMO
  ResType=$8027 rCursor, ID=$0000002,
  ResFile=$0FFF
...
```

The caret asks Nifty List to deref the handle that precedes it. The colon and the "h" ask Nifty List for a hex dump on the range of memory that the pointer pointed to by the handle points to.

Hehehe. I love a good indirection early in the morning.

See how easy this is? I can make excellent use of

Nifty List with just these two commands, "oink" and "caret". There are lots, lots more, of course.

You can get disassemblies of a range of memory just like (actually, better than) the monitor. You can get descriptions of commands with the equals sign (=i, for example, would provide a description of the info command we looked at earlier.) And you can extend Nifty List with command modules.

Taking Command

Oh yes, the command modules... David Lyons has created a very programmer-extensible environment. By writing an NDA-like set of routines, you, too, can design your own Nifty List commands. Pop 'em in the same folder Nifty List lives in and presto chango - instant added commands. This looks to be a promising avenue for a future 8/16 article - I hereby declare it to be on our wish list.

The two command modules included with the shareware package are BB (Big Brother), and Goodies. You can get a list of all the commands in all of the command modules available by typing "=\" or "?\".

The Best is Last

Nifty List is not only easy to use, it is easy to buy. It is a \$15 shareware product available from:

DAL Systems
P.O. Box 875
Cupertino, CA 95015

I have only scratched the very surface of Nifty List's nifty feature list. To better help you decide if it has features and/or abilities you need, Figure 2 contains a list of my favorite commands and their descriptions.

Nifty List is kinda like turning on a flashlight in a dark room. You can't see everything all at once, but it beats the heck out of banging around in the dark.



The Resource Dilemma

Resources are both a terrific opportunity and a dismal dilemma for GS programmers. No matter where you turn, somebody wants you to shell out \$100+ for the "perfect" package for working with these beasties. A Merlin owner without the APW/Orca shell is looking at spending over \$150 in order to use Rez, Apple's resource compiler (because you gotta have the shell to use it).

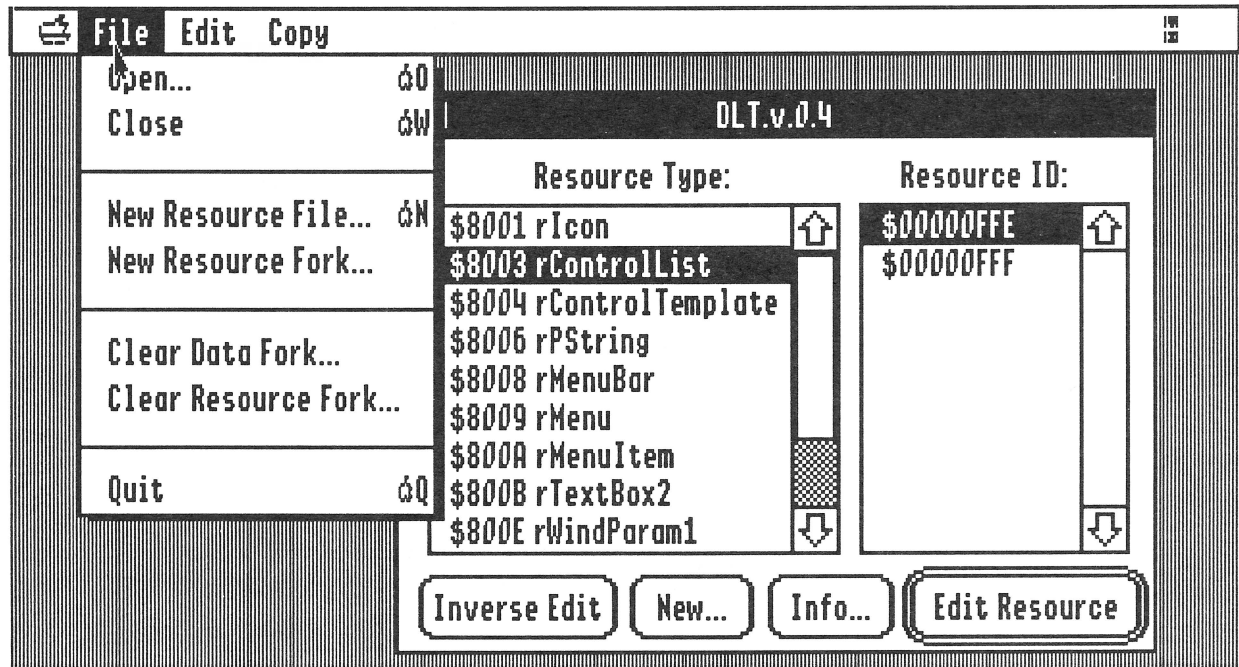
Don't get me wrong, I've fallen in love with both Orca and Rez. But the good ol' capitalist society in which we live is spawning more than one way to skin the resource cat.

Up until recently, Rez was the *only* mechanism around for cutting, copying, and/or pasting resource between resource files. Even Genesys and DesignMaster cannot do that yet. Thanks to shareware author Jason Coleman, that is no longer so.

Figure 2: The Best of Nifty List List

```
? .....displays a screen full of all the main commands
?x .....displays information about command "x". You can effectively use the ?
    and ?x sequence to avoid reading the docs for awhile.
=\ ... displays a list of all commands in the Nifty List modules available
_ .... evaluate toolbox call expression and display result (yes, you can
    do toolbox calls directly from Nifty List).
" .... displays information about parameters for a toolbox call. E.g.
    "NewHandle
: .... store data directly into a memory location (like the monitor)
L .... List (disassemble) machine code
M .... toggles 8 and 16 bit memory operations
X .... toggles 8 and 16 bit index operations for the list command
T .... prints the name and entry point for a tool or all tools in a set
H .... print info on a handle or on a group of handles of a certain type
W .... What handle - determines what handle an address belongs to.
```

Figure 1: LLRE's main window and File menu



Jason's LLRE (Low Level Resource Editor) is probably your quickest and cheapest path to resource creation. There's no syntax to learn, no big check to write, and no wait. It's here. It's now. It's happening.

It's no John Kennedy

I know a lot of folks are going to argue vociferously that LLRE doesn't even come close to the power and flexibility of Design Master, Genesys, or Rez. They are absolutely right. But I believe that most of you are a little leery of resources to begin with, and even more so when somebody wants you to shell out \$150 for the privilege of using them.

LLRE is a great introductory path. And even though I use both Rez and Genesys regularly, LLRE still gets called into action on virtually every job I do. The reason is that it copies resources from file to file in a faster and simpler fashion than Rez, and Genesys can't do that task at all.

Please note that Genesys (and probably Design Master) will undoubtedly acquire those capabilities in time. But LLRE gives it to you now and for very little expense (\$25 shareware).

Figure 1 shows LLRE's main window along with the Files menu. As you can see, you can create

a new resource file, put a resource fork on a file without one, and clear out either the resource or data forks of any file.

These are neat features, but beware: you can really mess things over in a hurry. If you ditch the data fork of an S16 application, for example, you've ditched the program code itself. I can hardly wait for Mac programmers to start destroying GS applications.¹

You'll also notice the main window; Jason has a very nice scrolling display of the resources in the current resource fork. In this case you're looking at the resource fork of our very own DLT (the S16 version). If you select a resource type (the left list), you'll be shown all the ID's of the all the individual resources of that type (the right list).

Figure 2 reveals LLRE's ability to copy entire resource forks or shuffle single resources. This is a very slick feature and is the primary reason that LLRE is useful here and now.

The reason why LLRE is not as robust as its commercial counterparts is readily apparent when you proceed to actually create a resource from scratch (e.g. an icon or menubar).

1. Macintosh programs live in CODE resources. The data fork of a Mac application rarely contains any actual program code.

On the plus side, you can import the actual data from another resource with the push of a button. On the minus side, if you're creating actual data, you've got to enter it in hex or ascii. This means that LLRE is pretty much okee dokee for text string type data (Pascal strings and string lists) and custom data types (those you define), but it is pretty miserable entering the data for a cursor or icon in hex.

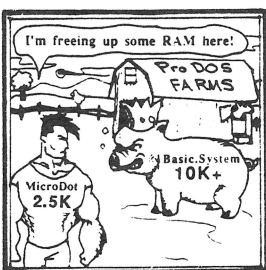
This is where Genesys and Design Master absolutely shine since they come with full blown editors for those kinds of items.

Still, LLRE is a powerful tool for your arsenal. From a marketing point of view, author Jason Coleman has taken a brilliant approach, having seen and filled a void quickly and efficiently. Let me repeat that, unless you have Rez, there is no other method for copying, cutting, and pasting resources between files or programs. And even if you do have Rez, LLRE may prove to be quicker and more intuitive for such tasks. As a \$25 shareware offering, it is well worth the price if you are heavy into the "resource-thing" on the IIGs.

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


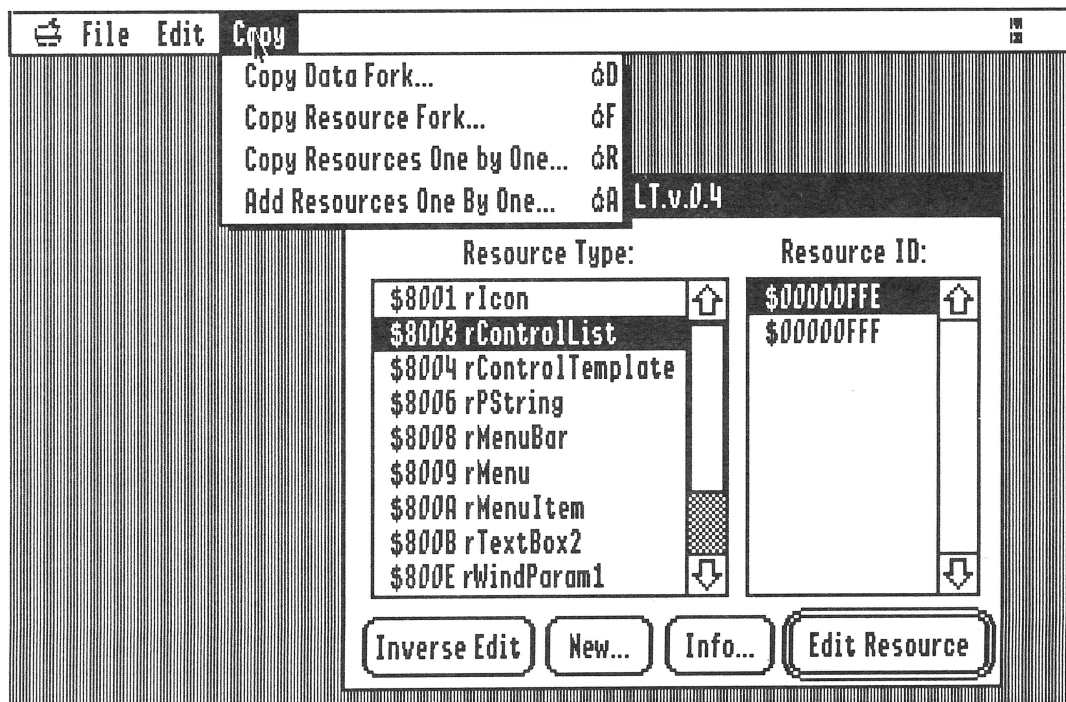


Figure 2 : LLRE's Copy menu





The ZBasic
Zealot

Miscellanea Month II

by Ross W. Lambert

My latest ventures into ZBasic's memory usage have proved enlightening, but I'm afraid I've not yet stumbled upon the variable space "pointer" that would permit us to store variables in the graphics pages directly. I've had a lot of letters wondering why I've gone to such lengths to shuffle data here and about when a simple POKE or two would probably reset things such that the space could be used automatically.

Well, folks, it might be possible. I don't know, and I've not had time to fully disassemble the darned thing yet. My hunch is that the 8K in graphics-land of main memory could be got, but aux mem is probably out of bounds (at least not without rewriting ZBasic's variable storage and lookup routines).

I'll keep poking around.

ZStatus Report

In the mean time, I'd like to deal with a couple of other common questions. First, what is the status of the Apple II ZBasic as far as Zedcor is concerned? The answer is simple: it is dead. They've not had an Apple II person in their employ since Greg Branche went to work for Apple, Inc. (nearly three years ago!). I think they could find somebody who'd take care of their users and pay them a royalty for the language sales (thereby taking on all their headaches and paying them for the privilege). But to my knowledge they've turned down every offer anyone has made (including ours). I don't think letters are of any consequence in Tucson, but you might give it a try if you want to continue using the language. I think they owe it to the current owners to try and find someone who'll

take care of them better. It is one thing to discontinue a product; it is another to tell your customers to take a hike.

ProTools II?

Second, what is the status of ProTools II? Again, the answer is simple: it ain't gonna happen. We're not working on it anymore. Instead, I plan to share the routines and functions we *did* develop within our pages. In short, we're giving it away (to 8/16 subscribers, anyway.)

We plan to support ZBasic in 8/16 as long as a sizable number of you are using it. My guess is

that we'll continue this column as a monthly for at least another year, and then semi-regularly thereafter. ZBasic is still, in my humble opinion, the premier 8 bit Apple II BASIC compiler (thank you, Greg Branche).

The distressing thing to me is that those of you leaving Z-Land are not dropping it in favor of another language, you're dropping the II, and not for the Macintosh.

Apple's got good things in the queue for the II (I've seen 'em with my own eyes). Let's hope it is not too little too late.

The ProTools II Folder

As I indicated, we're going to be dishing out ProTools II piecemeal herein. So let's start dishing it up.

"It is one thing to discontinue a product; it is another to tell your customers to take a hike."

Want some pi?

The first tidbits are two convenience functions, FN Angle2Radians and FN Radians2Angle.

QUICK NOW: How many radians in a circle? No fair looking it up in a book, either.

With these two functions you can program and think in "normal" angles and still use functions and routines that require radians.

Oh, by the way, there's 2 pi radians in a circle (i.e. 360 degrees).

The function does the conversions by using proportions, that is it finds how much of the circle you're talking about and takes that proportion of radians (or degrees if you're reversing the process).

The chart in Figure 1 shows how radians and degrees relate to each other around a circle.

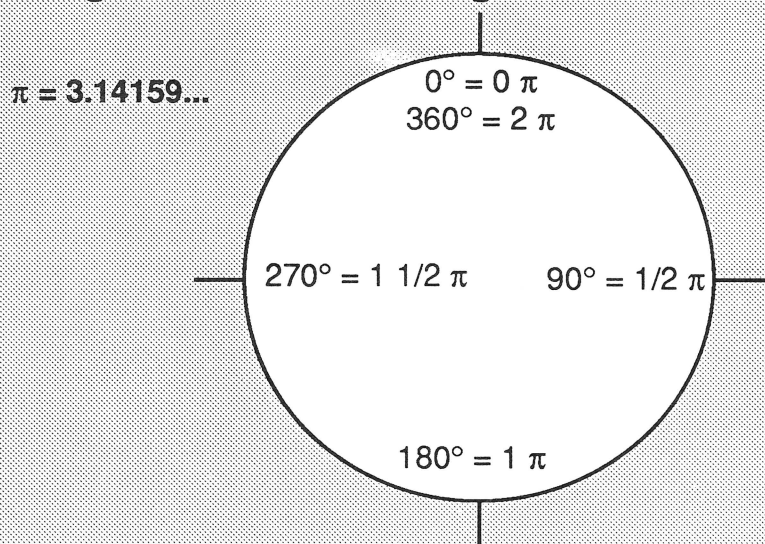
Listing 1: FN Angle2Radians and FN Radians2Angle

```

REM -----
REM FN Angle2Radians
REM FN Radians2Angle
REM These are public domain
REM -----
REM
REM DESCRIPTION: The Angle2Radians fn
REM will convert an angle (the unit of
REM measure most of us carbon types
REM understand) into radians (the
REM unit of measure used by most scientific
REM fns, etc.) Radians2Angle reverses the
REM procedure.
REM
REM VARIABLES: Angle - in degrees
REM              Radians - the result
REM -----
REM =====
REM
:
LONG FN Angle2Radians! (Angle!)
  Radians! = (2 * 3.14159 * Angle!)\360
END FN = Radians!
:

```

Figure 1: Radians and degrees



```

LONG FN Radians2Angle! (Radians!)
  Angle! = (Radians! * 360) \ (2 * 3.14159)
END FN = Angle!
:
:
REM =====
:
:
PRINT "Enter an angle: ";INPUT Angle!
Radians! = FN Angle2Radians! (Angle!)
PRINT "That equals "Radians!" radians."
Angle! = FN Radians2Angle! (Radians!)
PRINT "Converted back, that is an angle of
";Angle!;" degrees."
PRINT
PRINT "Press a key to stop..."
DO
  K$ = INKEY$
UNTIL LEN(K$)
END

```

"There's lies, damn lies, and statistics." - Mark Twain

Twain's dictum notwithstanding, one of our most requested functions has been FN Stats. Perhaps it is because a disproportionate number of educators are using ZBasic (perhaps due to its exceptionally high degree of numerical accuracy?). At any rate, the folks doing all this asking have written gradebook programs, psychology stats packages, etc., and have gotten good mileage out of this beastie already. Caveat emptor here: check my formulas!

FN Stats will examine a range of data in an array and return a wide array of statistical information to you, including the sum of all the items, the sum of the item's squares (useful for determining the variance and standard deviation), the mean (the arithmetic average), the median (the middle score), the mode (the score that occurred the most often), the highest and lowest scores, and the standard deviation (essentially the average difference between scores).

One slightly disconcerting thing: the function will report the lowest score as the mode if all the scores have occurred the same number of times. If it is important to have a valid mode, you need to scan the E(x) array to make certain that at least one score has occurred a different number of times than your modal score.

FN Stats assumes an unsorted data array, and therefore performs a QuickSort on it. This takes the bulk of the time and it is best if eliminated (if possible). Even with miscellaneous random data, the function is speedy enough for classroom sized chunks of data (about 6 seconds on my GS).

I don't feel too bad about the speed since my statistics prof had to tie up her micro for days when performing these same procedures on large samples. It is somewhat the nature of the beast, although I'd welcome any improvements y'all might want to send my way.

Listing 2: FN Stats

```

REM -----
REM FN Stats
REM This is public domain
REM
REM DESCRIPTION: This function will scan
REM any arbitrary range of data within an
REM array and return the mean, median,
REM computed mode, standard deviation
REM range, summation, and sum of squares.
REM It requires that you pass the starting
REM element number and total number of
REM elements to include.
REM
REM VARIABLES:
REM ARRAYSTART - the 1st element to use
REM TOTAL_ITEMS - the total number of
elements to use
REM Median# - the middle score
REM Mean# - arithmetic average
REM SumOfSquares# - just what it says (used
internally)
REM High# - the big score

```

```

REM Low# - the lowest score
REM Mode - datum which occurred the most
REM ST(X) - used in QuickSort routine
REM
REM -----
:
DIM DataItem#(999): REM up to 1000 data items
for this example
DIM E(999) : REM ...for working out mode
DIM ST(30,1) : REM ...for Quick Sort routine
REM
REM =====
REM The function... note that data must be
"pre-deposited" in DataItem#(X)
REM
:
LONG FN STATS (ARRAYSTART,TOTAL_ITEMS)
SUM#=0:REM init in case called more than
once
Low# = DataItem#(ARRAYSTART)
High# = DataItem#(ARRAYSTART)
SumOfSquares# = 0
:
REM scan data
:
FOR X = ARRAYSTART TO
ARRAYSTART+TOTAL_ITEMS-1
SUM# = SUM# + DataItem#(X)
SumOfSquares# = SumOfSquares# +
DataItem#(X)^2
IF DataItem#(X) < Low# THEN Low# =
DataItem#(X) :REM calculate range
IF DataItem#(X) > High# THEN High# =
DataItem#(X)
NEXT
:
:
StDev# = SQR((SumOfSquares# -
((SUM#^2)\TOTAL_ITEMS))\ (TOTAL_ITEMS-1))
:
"QUICK SORT"
SP=0: ST(0,0)=0:ST(0,1)=0
ST(0,1)= TOTAL_ITEMS-1
DO
L=ST(SP,0): R=ST(SP,1):SP=SP-1
DO
LI=L:RI=R:DataItem# = DataItem#((L+R)/2)
DO
WHILE DataItem#(LI)<DataItem#
LI=LI+1
WEND
WHILE DataItem#(RI)> DataItem#
RI=RI-1
WEND
LONG IF LI<=RI
SWAP DataItem#(LI), DataItem#(RI)
LI=LI+1:RI=RI-1
END IF
UNTIL LI>RI
LONG IF (R-LI)> (RI-L)

```

```

LONG IF L<RI
  SP=SP+1: ST(SP,0)=L: ST(SP,1)=RI
END IF
L=LI
XELSE
LONG IF LI<R
  SP=SP+1: ST(SP,0)=LI:ST(SP,1)=R
END IF
R=RI
END IF
UNTIL R<=L
UNTIL SP=-1
:
Mean# = SUM#\TOTAL_ITEMS
:
REM figure median
:
Midpoint = TOTAL_ITEMS/2.0
LONG IF Midpoint = INT (Midpoint) : REM do
we have even # of items?
  Median# = (DataItem#(Midpoint) +
DataItem#(Midpoint+1))/2.0 :REM yes, so avg
middle 2 items
XELSE : REM no
  Median# = DataItem#(Midpoint) :REM it's
odd, so take middle item
END IF
:
FOR I = 0 TO TOTAL_ITEMS-2
  FOR J = I + 1 TO TOTAL_ITEMS-1
    IF DataItem#(I) = DataItem#(J) THEN E(I)
= E(I) + 1
  NEXT
NEXT
:
FOR I = 0 TO TOTAL_ITEMS-1
  IF E(I) > NumTimes THEN NumTimes =
E(I):ActualMode = I
NEXT
:
END FN
:
REM =====
:
REM -----
REM Demo
REM -----

MODE 2
:
"Start"
TOTAL_ITEMS = 30 :REM classroom sized group
PRINT "Creating random numbers..."
RANDOM 12345 :REM initialize random # seed
FOR X = 0 TO TOTAL_ITEMS-1
  DataItem#(X) = RND (1000) :REM generate
random #s 1 to 999
NEXT
:
PRINT "Calculating statistics..."

```

```

FN STATS(0,TOTAL_ITEMS) :REM start
with 0th element and include all
:
:
PRINT:PRINT
PRINT "The sum of all the data: ";SUM#
PRINT " The sum of the squares:
";SumOfSquares#
PRINT " The mean: ";Mean#
PRINT " The median: ";Median#
PRINT " The mode:
";DataItem#(ActualMode)
PRINT " The standard deviation: ";StDev#
PRINT " The highest value: ";High#
PRINT " The lowest value: ";Low#
PRINT
INPUT "Press RETURN...";R$

```

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.
- Converts any size program.

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

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

Accessories 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
Conservor GS	\$89
A+ Optical Mouse ADB	\$87
Cordless Mouse ADB	\$109

GS Hardware

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

Software

Utilities

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

Vitesse Salvation Series:
Guardian— HD Backup \$29
Renaissance— Optimizer \$29
Exorciser— Virus Detector \$26

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

Business

AppleWorks GS	\$212
Manzanita Businessworks	\$294

Education

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

GS Numerics

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

Zip GS 8 Mhz \$269

Entertainment

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

GS Starter System

- Apple IIGS 1 Meg CPU,
keyboard and mouse
- Magnavox RGB Monitor
- Fortris ImageWriter
compatible printer
- AMR 3.5" Drive
- Mouse pad
- Box of 10 Maxell 3.5" Disks
\$1599

GS Power System

- Apple IIGS 1 Meg CPU,
keyboard and mouse
- Apple Color RGB Monitor
- Apple ImageWriter II with
32k buffer
- Apple High Speed DMA SCSI
- AMR 40 Meg GS Partener HD
- Chinook RAM 4000 w/ 2 Meg
- AMR 3.5" Drive
- Mouse pad
- Box of 10 Maxell 3.5" Disks
\$2959

Modems

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

Hard Drives

Chinook CT100 16k cache	\$780
UniStore 80 Meg HS HD	\$529
UniStore 60 Meg HS HD	\$474
AMR GS Partner (0 Footprint)	
40 Meg	\$420
60 Meg	\$640
80 Meg	\$700
100 Meg	\$876
AMR 45 Removable HD	\$769
CMS 60 Meg HD	\$539
Apple DMA SCSI w/ purchase of HD: \$96 Without: \$101	

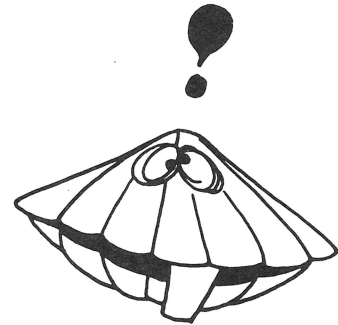
All HDs come formatted w/ GSOS or
Mac system software, and 5-10 megs of
PD, Share/Freeware, NDAs, CDAs, and
INITs.

InnerExpress \$85

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

To Shell With It

by Morgan Davis



Okay, let's all set our phasors to "stun" and point them in the general direction of the APW C standard library file.

Sadly, APW C programs requiring command line arguments cannot be used from any shell other than ORCA/M (aka APW). The myriad of programs written in APW C are reserved for those who purchased Apple's APW or The Byte Works' ORCA/M shell. Users of ProSel-16, ECP-16, or other third party environments are left out of this shell game because of one silly misfeature of the APW C standard library. And it doesn't look like Apple is about to upgrade the compiler any time soon to provide this courtesy, let alone fix more serious bugs.

So, how do we write APW C programs that run under different environments? The answer: scrap the START.ROOT object file that is linked into all APW C programs and use our own version. Not only is this simple to do, but it also shrinks the size of your C programs down significantly.

In addition, applications (S16 files) can be created with APW C that will prompt the user for a command line so that C applications requiring command line arguments can even be run from the Finder.

The trick is to create a new START.ROOT. Now, before you drag your old START.ROOT into the trash can, sending it into oblivion, just rename it. You may need it later. I've renamed mine to SHELL.ROOT. You'll find START.ROOT in prefix 2/, the LIBRARIES directory.

By the way, whenever you see ORCA/M shell used here, it includes the APW shell as well.

The Problem

It is an understood policy that all programming shells for the Apple IIGS put an eight-byte

identification string at the beginning of any command line passed to a shell (EXE) program. The ORCA/M shell uses the string, BYTEWRKS, not surprisingly.

Programs written in APW C that use command line arguments actually compare the shell's eight-byte identifier to BYTEWRKS, and if it isn't there, the C program concludes, "Well, it ain't ORCA/M, so I'll just assume there are no arguments." We all know what happens when you assume.

A New Start

Figure 1 shows an ORCA/M assembly listing of our new START.ROOT. Briefly, it performs the following functions:

- Sets up `_ownerid`
- Sets the IIGS's data bank register to access our globals
- Creates a pointer to a command line
- If a command line is not present, the text devices are initialized
- Performs a long jump to the `main()` C function
- When `main()` returns, decides to quit via an RTL or GS/OS

"APW C programs requiring command line arguments cannot be used from any shell other than ORCA/M (aka APW)."

QUIT

There's quite a bit of magic here. But before we get into it, it is important to understand how shell (EXE type) files are launched.

When control is passed to our START.ROOT code at the beginning of a C program, the accumulator holds the program's ID assigned to it by the launching application or operating system. The X and Y registers, if both are not zero, point to a special text buffer that holds the command line arguments. As mentioned, the command line text buffer begins with an eight-byte shell identification string. Our START.ROOT saves the command line pointer

in `_cmdLine`.

If the C program's file type is S16, it is launched as a regular application, and no command line is provided, so X and Y are both zero. If the type is EXE, a command line is always provided, even if no arguments are given by the user.

START.ROOT automatically configures itself for quitting depending on the presence of a command line. If no command line is available (meaning the program has an S16 type assigned to it), the application quits via a GS/OS Class 0 QUIT call. If the program is a standard EXE type, launched from a shell, it will quit by executing an RTL instruction. This is where all the fiddling with the `_xQuit` variable comes into play. Upon returning from `main()`, `_xQuit` is either zero or \$0029.

The text screen and keyboard are initialized only when the program is launched from a non-shell launcher, like the Finder. When used under a shell, the text I/O devices should not be altered.

The mechanics used here are hardly high tech. The procedure is simple, as is this whole fiasco. But, now the fun begins: building the new START.ROOT.

It is best to create an ORCA/M shell script as shown in Figure 2 to generate START.ROOT. Execute the script that contains these instructions, and a new START.ROOT is deposited into prefix 2/ (the LIBRARIES directory).

Just Say No To Shell Commands

A few rules must be followed in order to make use of the new START.ROOT. First of all, the C program cannot call any shell-dependent functions. After all, the object of this is to create C programs that are shell-independent. This means you can't call `INIT_WILDCARD()`, `NEXT_WILDCARD()`, `STOP()`, and other shell-specific functions. If you need these, you can write equivalent functions on your own.

Second, our START.ROOT does not start up the SANE toolset. In fact, it doesn't start any toolsets. If your C program uses floating point numbers, it is your responsibility to get SANE, and any other toolsets your program requires, started up at the beginning of your program. This can be done from within `main()`.

Finally, don't use the `exit()` function. You can

simulate it, however, by setting `xStatus`, one of START.ROOT's variables. Do this just before `main()` returns. Yes, this means that your programs must gracefully return from `main()`, as all well-behaved programs should. If a program must terminate from outside of `main()`, creative use of `setjmp()` can be applied.

ARGS.C: A Sample

Our new START.ROOT provides the foundation for new shell-independent C programs. On the C side of things, only a few supporting functions are needed to get command line arguments into our programs. The example listing in Figure 3 shows a complete C program, called ARGS.C. This example is an excellent template to follow when creating your own shell-independent C programs.

ARGS.C begins by defining these constants:

- PROGRAM: Title as shown in the "usage" part of your program
- COPYRIGHT: Your copyright notice
- PROGNAME: The intended file name of your program
- ARGUMENTS: Synopsis of the arguments your program requires
- ARGV_MAX: Maximum number of arguments your program might use

Following these definitions comes the inclusion of header files that ARGS.C requires.

Next, the program declares external references to some of START.ROOT's variables:

<code>_cmdLine</code>	Pointer to the shell's command line (or NULL)
<code>_xStatus</code>	Exit status variable (default value is zero)

Now, the next three functions make up the guts of ARGS:

ShowUsage	Displays program title and usage information
GetInput	A routine to get a line of input from the user

`ccommand` Parses the command line,
returning the argument count

Clearly, `ccommand()` is the heart of the argument processing system. By making a call to `ccommand()`, a C program can gather all the necessary information needed to set up the `argc` and `argv[]` variables that C programs use.

How `ccommand()` Works

Notice how `ccommand()` is called from within `main()`. Inside `main()`, the `argc` and `argv[]` variables are declared as local variables, not formal parameters to `main()`. `argv[]` is an array of character pointers and will hold pointers into the command line where each argument begins. `Argc` will hold the count of the number of arguments on the command line, including the name of the C program itself. When `ccommand()` is called, the address of the `argv[]` array is passed.

First, `ccommand()` determines if a command line is provided by the shell. If no command line is present, `ccommand()` calls `ShowUsage()` to display the program's title and usage information. It then prepares to make the `GetInput()` call in order to obtain a line of input from the user. This occurs if `ARGS` has an `S16` file type and is launched as an application.

Finally, `ccommand()` chews on the command line, setting up the `argv[]` array of pointers, and returning the argument count. The parsing simply involves locating the beginning of each argument by skipping any leading or trailing space characters. No provision is made for parsing quoted strings (with or without embedded spaces), handling character escapes, nor detecting I/O redirection requests when the program is launched as an application. These features are left for you to add later, should you need them.

Compiling and Running `ARGS`

To compile and link `ARGS.C`, just type:

```
cmpl args.c keep=args
```

...just as you would with any ordinary APW C program.

After it has been successfully compiled, type "args" alone at your shell prompt. If the program is working correctly, it displays:

```
Args (start.root demo) 1.0 30-Jun-90
Copyright (C) 1990 Morgan Davis Group
```

```
Usage: args [ arguments... ]
```

Now type:

```
args testing one two three
```

And, the following is displayed:

```
args testing one two three
```

It works! Try it again adding extra spaces between arguments to see the results. If you haven't guessed, `ARGS` displays the command line arguments from `argv[0]` to `argv[argc-1]`.

Application Test

Now, here's the big test. Change `ARGS`'s file type from `EXE` to `S16` using the `FILETYPE` command:

```
filetype args S16
```

...and launch it by typing in `ARGS`. (You could enter some arguments, but because `ARGS` is no longer a shell program, the arguments are just ignored). The screen clears, the program's title and usage information is shown, and you're prompted to enter a command line. Here is where the `GetInput()` function comes into play.

After entering some arguments and pressing `RETURN`, the argument list is displayed just as it was when ran from a shell. Plus, the program is courteous enough to ask you to press a key before it erases the screen and quits to `GS/OS`.

Neat. Now you can write C programs that use command line arguments from shells other than `ORCA/M`, and even from non-shell program launchers like the `Finder!`

Enhancements

Fortunately, `START.ROOT` and the accompanying C functions offer a lot of flexibility. For example, rather than obtaining a line of input from the text mode, the C program might want to bring up a nice dialog box in the super hires mode. Additionally, here are some other "tweaks" you may want to try:

`_xQuit`. When the program is run as an `S16`

application, change the value of this integer to \$2029 to use a Class 1 GS/OS Quit call instead of a Class 0 QUIT.

`_xQPrompt`. Normally, this Boolean variable is set to 1, which causes the C program to ask [Any Key] before quitting when run in application mode. Setting this to zero will bypass the prompting.

`qPath`. This points to a file name to launch when `main()` quits. Normally, this doesn't point to any file name, so a standard Quit takes place, returning you to the launching program. Set this to point to any S16, EXE, or SYS file name to quit to a different application.

`qFlag`. This integer holds the flags for use with the GS/OS QUIT function. Diddle the bits in this variable to alter the way your program quits.

The new START.ROOT gives you just enough to get most any C program off the ground, yet saves a lot of disk space and memory by not assuming that a C program requires stuff that it may never use.

About the author:

Morgan Davis is founder of the Morgan Davis Group, not affiliated with the Morgan Davis Band of Canada. However, Morgan would proudly wear a Morgan Davis Band T-shirt if given one. He hopes to make regular contributions to 8/16.

Listing 1: START.ASM Source

```
*****
***
*** start.asm Source for a better START.ROOT
*** for use with APW C programs
***
*** Copyright (C) 1990 Morgan Davis Group
*** Most Rights Reserved

        case    on        ; case sensitive for C
        objcase on

_start  start  main ;start in "main" load seg
        using  ~globals
        pha          ; save _ownerid for now
        lda      #_toolErr|-16; make ~globals
segment...
        xba          ; ...where the DBR references
        pha
```

```
plb
plb
pla          ;retrieve _ownerid
sta      |_ownerid ;store copy _ownerid
sty      |_cmdLine ;save cmd line ptr
stx      |_cmdLine+2
txa          ; prep for NULL comparison
ldx      #0          ; X = 0 (quit flag)
ora      |_cmdLine  ; a command line?
bne      |_doMain   ; yes! Don't init
          ;text screen
jsr      txtinit ;init text I/O devices
ldx      #$0029    ; set class 0 quit
          ; call number
_doMain  stx      |_xQuit ; set up xQuit code
        jsr      main   ; call the program
        lda      |_xQuit ; exit type?
        bne      |_pquitx ; go ProDOS

        lda      |_xStatus ; return via RTL
        rtl

_pquitx  sta      |_osqnum ; Save quit code num
        lda      |_xQPrompt;prompt before quit?
        beq      |_osquit ; no

        pea      |_xAnyKey|-16 ;"Any Key" prompt
        pea      |_xAnyKey
        ldx      #$200C          ; WriteCString
        jsr      $E10000

        pha          ;now get a keypress
        pea      0
        ldx      #$220C          ;ReadChar
        jsr      $E10000

        pla          ;write a newline
        pea      |_xAKNull|-16
        pea      |_xAKNull
        ldx      #$1A0C          ; WriteLine
        jsr      $E10000

        jsr      txtinit ; init text I/O
          ;devices (clear scrn)
_osquit  lda      |_xStatus ;just in case it
          ;might be used
        jsr      $e100a8 ;call operating system
_osqnum  dc      i2'$0029' ; QUIT
        dc      i4'qPath'

txtinit  pea      1
        pea      0
        pea      3
        ldx      #F0FC ;SetInputDevice (1, 3L)
        jsr      $E10000

        pea      0
        ldx      #$150C          ; InitTextDev (0)
        jsr      $E10000
```

```

                                unset echo
pea    $007f
pea    $0000
ldx    #$090C          ; SetInGlobals
                                ; (0x007f, 0x0000)
jsl    $E10000

pea    1
pea    0
pea    3
ldx    #$100C;SetOutputDevice (1, 3L)
jsl    $E10000

pea    1
ldx    #$150C          ;InitTextDev (1)
jsl    $E10000

pea    $00FF
pea    $0080
ldx    #$0A0C          ; SetOutGlobals
                                ; (0x00FF, 0x0080)
jsl    $E10000
rts
end

~globals data ~globals ; ~globals segment
_toolErr entry
    ds    2          ; tool/disk error results
_ownerid entry
    ds    2          ; ID of program
_cmdLine entry
    ds    4          ; command line pointer
qPath  entry
    ds    4          ; path pointer to next app
qFlag  entry
    dc    h'0000'          ; quit flags
_xQuit entry
    dc    h'0029'          ; quit command number
                                ; ($0000, $0029, or $2029)
_xStatus entry
    ds    2          ; return status for shell
_xQPrompt entry
    dc    h'0001'          ; Boolean: prompt on
                                ;app exit
_xAnyKey dc    c"[Any Key]"; exit prompt str
_xAKNull dc    h'0000'; end of prompt string
                                ;/ newline text

end

```

Listing 2: START.ROOT Creation Script

```

set echo 1
assemble start.asm keep=start
crunchiigs start
delete start.a start.root
move -c start.obj 2/start.root

```

Listing 3: ARGS.C

```

/*****
***
*** args.c      A program to demonstrate
***            shell-independent APW C programs
***
*** Copyright (C) 1989-1990 Morgan Davis Group
***
*****/

#define PROGRAM "\pArgs (start.root demo) 1.0
30-Jun-90"
#define COPYRIGHT "\pCopyright (C) 1990
Morgan Davis Group"
#define PROGNAME "args"
#define ARGUMENTS "\p [ arguments... ]"
#define ARGV_MAX 50

#include <types.h>
#include <ctype.h>
#include <string.h>
#include <texttool.h>

extern ptr _cmdLine;
extern int _xStatus;

void
ShowUsage (name)
    char *name;
{
    ErrWriteLine (PROGRAM);
    ErrWriteLine (COPYRIGHT);
    ErrWriteLine ("");
    ErrWriteCString ("Usage: ");
    ErrWriteCString (name);
    ErrWriteLine (ARGUMENTS);
}

word
GetInput (prompt, stuffer, buf, size)
    char *prompt, *stuffer, *buf;
    word size;
{
    char *p = buf;
    word n = 0;
    int c;

    if (prompt)
        ErrWriteCString (prompt);

    if (stuffer) {
        strncpy (buf, stuffer, size);
        p += (n = strlen (buf));
        ErrWriteCString (buf);
    }
}

```

```

do {
    ErrWriteChar(5);          /* Cursor on */
    c = (ReadChar(noEcho) & 0x007f);
    ErrWriteChar(6);        /* Cursor off */

    switch(c) {
    case 8:
    case 127:
        if (n) {
            --p;
            --n;
            ErrWriteString("\p\b\b");
        }
        break;
    case 27:
        p = buf;
        n = 0;
    case 13:
    case 10:
        *p = 0;
        c = -1;
        break;
    default:
        if (isprint(c) && (n < size)) {
            ErrWriteChar(*p++ = c);
            ++n;
        }
    }
} while (c != -1);
ErrWriteLine("");
return (n);
}

word
ccommand(argv)
    char **argv;
{
    static char clbuf[255];
    word n = 0;
    char *cp = _cmdLine + 8;      /* skip
over shell's ID */

    if (!_cmdLine) {
        ShowUsage (PROGNAME);
        strcpy (clbuf, PROGNAME);
        strcat (clbuf, " ");
        GetInput ("\n# ", clbuf, clbuf,
sizeof (clbuf));
        cp = clbuf;
    }

    while (*cp && (n < ARGV_MAX)) {
        while (isspace(*cp))
            cp++;
        if (*cp) {
            argv[n] = cp;
            while (!(isspace(*cp)) && *cp)
                cp++;

```

```

        *cp++ = 0;
        n++;
    }
}

void
main()
{
    char *argv[ARGV_MAX];
    word argc;
    word i;

    argc = ccommand(argv);      /* get
arguments */

    if (argc == 1) {           /* if
none, show usage */
        ShowUsage (argv[0]);
        _xStatus = -1;
    } else {
        for (i = 0; i < argc; i++) { /* else
show arguments */
            WriteCString (argv[i]);
            WriteChar (' ');
        }
        WriteLine ("");
        _xStatus = 0;
    }
}

```

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

GENESYS



GS SAUCE RAM CARD

Now available and shipping!

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

Genesys is the first Apple IIGS CASE tool of its kind with an open-ended architecture, allowing for support of new resource types as Apple Computer releases them by simply copying additional Genesys Editors to a folder. Experienced programmers will appreciate the ability to create their own style of Genesys Editors, useful for private resource creation and maintenance. And Genesys generates fully commented source code for ANY language supporting System 5.0. Using the Genesys Source Code Generation Language (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

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**

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 compatible
- 100% GS/OS 5.0 and ProDOS 8 & 16 compatible
- 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% compatible 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:

<u>Apple //gs model</u>	<u>add these:</u>	<u>total GS RAM</u>
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:

OK	\$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 get a coupon to purchase GS Sauce for:

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

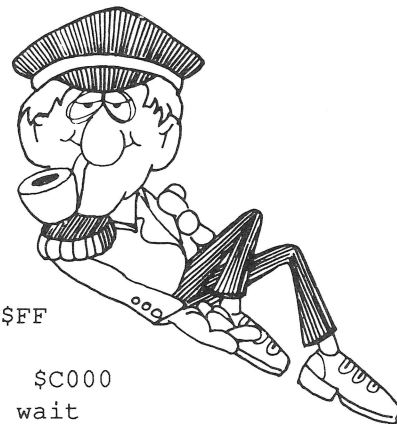
We hope you will see what an excellent value the GS Sauce card is: low power consumption, SIMMs technology, inexpensive, made in USA and lifetime warranty!

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



Generic Shutdown

by Jerry Kindall, Classic Apple Editor



Last month I presented a "front end" for 8-bit SYS applications and promised to be back this month with "back end" routines. Well, here they are. Shutting down an 8-bit application is much simpler than starting it up. You basically just close any open files, reset the display to a standard state, clean up any other messes you may have made (such as shutting down any interrupt handlers you've installed, and do a ProDOS quit call. No hay problema.

The routines in this article also handle the more complex procedure of quitting one program and starting up another without returning the user to his program selector. Occasionally you may want to present the user with an option to exit to BASIC (launching BASIC.SYSTEM), or to run the GS/OS Installer, or whatever. (To do the latter you naturally have to boot GS/OS before running your ProDOS 8 program.) Or you might just have a large program that you have segmented into multiple SYS files which must be run in a particular order.

To use the quit routine, simply include the SHUTDOWN file (using Merlin's PUT directive) anywhere convenient in your source, then include a "jmp quit" at the point you want the program to quit. To use the launch routine, use "jmp launch" instead. In either case, you will need to define the labels "runpfx" and "runpath" as the labels of Pascal strings containing the prefix and pathname of the application to be run. (Even if you don't use the launch routine, these labels must be defined for the file to assemble without errors, since the launch routines are assembled whether you actually use them or not.) Here's an example routine that waits for a keypress, then runs BASIC.SYSTEM from /HARD1:

```

4          typ    $FF
5
6  wait    lda    $C000
7          bpl    wait
8          lda    $C010
9          jmp    launch
10
11  runpfx  str    '/hard1'
12  runpath str    'basic.system'
13
14          put    shutdown

```

In most cases, you would not want to hard-code the prefix to a specific volume name as we have done here. You'd want to allow the user to select the volume, or get the volume name from the application directory as determined by the generic startup routine. Hard-coding the pathname of the next program to be run is OK in cases such as the one we have here, where presumably the user would be able to select a menu item which said something like "exit to BASIC".

You might be wondering how we launch 16-bit (GS/OS) programs from ProDOS 8. It's easier than you think -- easier, in fact, than launching ProDOS 8 programs! When

you launch a ProDOS 8 program from GS/OS, GS/OS patches ProDOS 8 in several places. One of the patches allows ProDOS 8 programs to perform what is known as an "extended quit". A regular ProDOS 8 quit parameter list looks like this:

```

dfb    $04          ;4 parms in list
dfb    $00
dw     $0000
dfb    $00
dw     $0000

```

An extended quit parameter list looks like this:

```

1  * Demo of shutdown routines
2
3          org    $2000

```

"You might be wondering how we launch 16-bit (GS/OS) programs from ProDOS 8. It's easier than you think..."


```

dfb $04      ;4 parms in list
dfb $EE      ;EE = Extended
dw  path     ;ptr to pathname
           ;of file to launch

dfb $00
dw  $0000

```

All you need to do is issue an extended quit, and GS/OS will get control, automatically launching the program you specify. GS/OS can, naturally, launch either 16-bit or 8-bit applications. If GS/OS isn't available, the extended quit call simply quits, which is what we want to do when we can't launch the desired application, anyway.

Although GS/OS can launch 8-bit programs, we can't count on GS/OS being available (or even on the machine being a IIgs, of course!), so we have to include our own code to launch SYS files. ProDOS 8 does not have an MLI command to launch a SYS file, so our routine must open the file, read it in, close the file, and JMP to \$2000 to begin execution of the program. Since we'll overwrite the memory from \$2000 up when reading the file, the code that reads the file into memory must live at a lower address. I picked \$1000. Looked at in this light, the launch routine probably becomes a little clearer.

As with the generic startup routines, I attempted to use as few global labels as possible. The ones I used were:

- shutdown - close all open files and reset display
- quit - call shutdown, then execute ProDOS quit
- launch - call shutdown, then launch P8 or GS/OS application
- dispatch, dstart, dend - used by the launch routine

I can think of only one enhancement that could be made to this routine: the ability to pass a startup path to another application via the launch routine. I didn't include this because it's something that probably won't be used very frequently. You might also want to break out the quit routine into a separate PUT file, to avoid including all the launch code in programs that don't use it.

```

1 *-----
2 * 8-bit Generic Shutdown Routines
3 * by Jerry Kindall
4 * 8/16 - September 1990

```

```

5 *-----
6         lst  off
7
8 *-----
9 * Program Shutdown
10 * This routine called by QUIT and LAUNCH
11 * to close all open files, reset display,
12 * & do all manner of things necessary to
13 * exit program by a ProDOS QUIT call or
14 * by launching another program. Add code
15 * necessary to shut down your specific
16 * application.
17 *-----
18
19 shutdown lda #0
20         sta $BF94      ;level
21         jsr $BF00      ;close all files
22         dfb $CC
23         dw  :pclose
24
25         jsr $FC58      ;home
26
27         lda $BF98      ;80-columns?
28         and #$02
29         beq :initvid   ;nope
30         lda #$15      ;turn it off
31         jsr $C300
32
33 :initvid jsr $FE89      ;initkbd
34         jsr $FE93      ;initvid
35         jsr $FE84      ;setnorm
36         jsr $FB39      ;settxt
37         bit $C054      ;page 1 text
38
39         lda #0;clear ProDOS memory map
40         ldx #$17
41 :clrmap  sta $BF58,x    ;ProDOS bitmap
42         dex
43         bne :clrmap
44         lda #%11001111 ;except 0,1,4-7
45         sta $BF58
46         lda #%00011111 ;pages $B3-$B7
47         sta $BF6E      ;bitmap+$16
48         lda #%11100001;pp $B8-$BA; $BF
49         sta $BF6F      ;bitmap+$17
50         rts
51
52 * MLI parmlists for shutdown routine
53
54 :pclose  dfb $01
55         dfb $00
56
57 *-----
58 * Quit the application
59 * Shuts down program then exits
60 *-----
61
62 quit    jsr  shutdown
63         jsr  $BF00
64         dfb  $65

```

```

65          dw  :pquit
66          jmp  quit ;never executed (we
hope)
67
68 * MLI parmlist for quit routine
69
70 :pquit    dfb  $04
71          dfb  $00
72          dw   $0000
73          dfb  $00
74          dw   $0000
75
76 *-----
77 * Launch another application
78 * Shuts down then exits by running
79 * another SYS or S16 program
80 *-----
81
82 launch   jsr  shutdown
83          jsr  $BF00 ;set pfx to next app
84          dfb  $C6          ;set_prefix
85          dw   :ppfx
86          bcs  quit
87          jsr  $BF00
88          dfb  $C4          ;get_file_info
89          dw   :pinfo
90          bcs  quit ;on error, quit
91          lda  :pinfo+4
92          cmp  #$FF          ;SYS file?
93          beq  :sys          ;launch 8-bit app
94          cmp  #$B3          ;S16 file?
95          bne  quit ;exit w/ normal quit
96          jsr  $BF00
97          dfb  $65          ;quit call
98          dw   :pequit      ;parms for
extended quit
99          bcs  quit ;on error, quit
100
101 :sys     ldx  #dend          ;copy
dispatcher code
102 :copy    lda  dispatch-1,x ; to $1000
103          sta  $FFF,x
104          dex
105          bne  :copy
106          jsr  $BF00
107          dfb  $C8          ;open call
108          dw   :popen
109          bcs  quit
110          lda  :popen+5 ;global close so
111          cmp  #1 ;ref should be 1, but
112          bne  quit ;this just in case
113
114          jmp  $1000 ;jump to dispatcher
115
116 * MLI parmlists for launch routine
117
118 :pequit  dfb  $04;extnd quit prmlst
119          dfb  $EE;flag extnded quit
120          dw   runpath ;addr of path
121          dfb  $00

122          dw   $0000
123
124 :popen   dfb  $03          ;open parmlist
125          dw   runpath      ;pathname
126          dw   $1C00        ;disk buffer
127          dfb  $00          ;ref num
128
129 :pinfo   dfb  $0A ;get_file_info
parmlist
130          dw   runpath      ;pathname
131          dfb  $00          ;access bits
132          dfb  $00          ;file type
133          dw   $0000        ;aux type
134          dfb  $01          ;storage type
135          dw   $0000        ;blocks used
136          dw   $0000        ;date mod
137          dw   $0000        ;time mod
138          dw   $0000        ;date created
139          dw   $0000        ;time created
140
141 :ppfx    dfb  $01          ;set_prefix
parmlist
142          dw   runpfx       ;pathname
143
144 dispatch = *;where dispatcher is
145             ;before move to $1000
146
147          org  $1000
148
149*Load and execute next SYS file.NOTE:
150*This runs at $1000. If it ran w/in
151*your app's normal memory, it could
152*be overwritten by prog being loaded!
153*When this gets control, next app's
154*file is already open & ref num is
155*known to be 1. All we must do is rd
156*the file, close it, & jump to $2000.
157* On err, executenormal ProDOS quit.
158
159 dstart   jsr  $BF00 ;rd data from
file
160          dfb  $CA          ;read
161          dw   :pread
162          php          ;save READ status
163          jsr  $BF00
164          dfb  $CC ;close
165          dw   :pclose
166          plp          ;get READ status
167          bcs  :quit ;err, do normal
bye
168
169          ldx  #$FF          ;init stack ptr
170          txs
171          jmp  $2000        ;enter next
app
172
173 :quit    jsr  $BF00
174          dfb  $65          ;quit
175          dw   :pquit
176          jmp  :quit

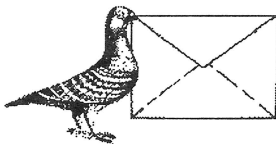
```

```

177
178 * MLI parmlists used by dispatcher
179
180 :pread   dfb   $04       ;read parmlist
181         dfb   $01       ;ref number
182         dw    $2000     ;address
183         dw    $9F00     ;bytes requested
184         dw    $0000     ;bytes read
185
186 :pquit   dfb   $04       ;quit parmlist
187         dfb   $00
188         dw    $0000
189         dfb   $00
190         dw    $0000
191
192 :pclose  dfb   $01       ;close parmlist
193         dfb   $00
194
195 dend    =    *
196
197 * Return assembly counter to correct addr
198
199         org   dispatch-dstart+dend
200         lst   on

```

Letters...



Is Late Better Than Never?

Dear Ross,

...The situation at the moment is that the June issue arrived (first class) on 12th June (postmarked 1st June), and the other outstanding issue (May) arrived on the 14th (postmarked 26th April)...

Some comments first about *8/16*. Generally, more than half the content is of direct interest to me. It is by and large well written, relevant and to the point. For those members of our

local user group like myself who are feeling the loss of *CALL A.P.P.L.E.*, *8/16* seems to be the last refuge of the serious programmer. Contributors such as Cecil Fretwell, who must be suffering from itchy pens and have migrated to *8/16*, make the publication potentially even more attractive.

The Apple II world needs *8/16*. I need *8/16*. Others around have told me they need *8/16*. However, for a magazine that proposes to be timely and up-to-date, we consider the delay and uncertainty of sea mail to be unacceptable. The only other publication that I get from the States is *nibble*, which costs \$90 per annum, airmail. \$45 per annum for *8/16* is in about the right vicinity for price, at its present stage of development.

So what are our options? I noticed with alarm that the postage on the June issue was \$4.32. I hope that's not the regular rate, but just a one-off. (From other post mailed from the States, I would have expected the post to be under \$2.00) Clearly, airmail postage at that rate is not compatible with your current charge of \$45 for non-North American subscribers. So how can we get airmail delivery at reduced costs?

One answer might be to follow A2-Central's example and use lighter paper. The other might be to produce a physically smaller magazine, by using a smaller typeface and a more compact layout (based on issues to date there is some room for an increase in the black/white ratio). But this may not please your domestic subscribers, who seem to prefer the current layout.

Another possibility is to take only disk subscription. We miss out on your art (and probably some of the ads, though they are mainly text and could probably be included), but I'm sure that the airmail postage on a 3.5" disk is cheaper than on the magazine... Perhaps you have other possibilities in mind. But I do stress that we place high value here on getting our monthly information "fix" with minimal delay.

There are a few others here who have seen the magazine and are considering subscribing but are waiting to see how the delivery situation resolves itself... What can you offer regarding delivery of the printed copy?

Sorry to have to expend most of my letter on such mundane things. I should be writing about some of the things I have found about assembly language when using Sourceror to poke about in other people's programs (like AppleWorks, picking up where Bob Sander-

Cederlof left off?). On the other hand, I would personally enjoy a bit of coverage of the "S" in IIGs (sampling, MIDI, etc.).

For now, best regards...

Yours sincerely,

John D. Smyth
Blackburn, Victoria Australia

John,

First of all, I'd like to thank you for your kind comments and for being so aware of the financial situation on the postage front.

\$4.32 is indeed typical for a 48 page+ magazine shipped first class from the US to any destination other than Canada or Mexico. I'm afraid that alone is the reason for nibble's \$90 price tag to you.

As for changing our format - such a move would cost us as many or more subscriptions than it would gain. We constantly receive letters requesting more graphics and a "classier" feel to the publication. There are even those requesting glossy pages throughout, a move that would nearly double the weight. Keep in mind that A2-Central is an eight page newsletter, a far different beast than a full-blown magazine.

We have not offered a first class foreign subscription rate because it would require so much special handling - a separate data base report as well as extra time metering the packages. However, we think we can handle the extra duties for a one year first class non-North American subscription for \$84.95. I know that is pricey, but the spreadsheet doesn't lie (most of the time, anyway). We will at least be able to offer this to those that can/will pay for it. And it is \$5.00 US cheaper than nibble.

However, I have an idea wherein y'all may be able to help yourselves and us, too. We are in the process of getting 8/16 into as many bookstores and distributorships as possible. It is slow going, let me tell ya. But if you could find an Australian periodical distributor who'd want to handle us (at least 20 or more copies per month), the per issue cost for shipping would drop dramatically.

As for your article suggestions... according to our legal counsel, we cannot run anything approaching a disassembly of a commercial product. It is, under US law, anyway, a blatant infringement of

copyright. As enjoyable as Bob's disassembly work with AppleWorks was (and his improvements on the code), we absolutely cannot follow suit or we risk having Apple Legal come roaring after us.

I'd love some MIDI and sound articles, too. For some reason we've not had a single submission in that category. I will declare them to be on our "wishlist" and try to hunt some down.

Thanks again for your thoughtful and well-written letter.

== Ross ==

DLT 0.4 Bug, Fast SCSI Quirks, & TEPaintText Fix from France

Dear Ross,

... I found some problems with DLT.04. First there is no Page Setup menu and so, we can't choose 'compressed', also called Macintosh printing mode. The 'cut, copy, and paste' menus are enabled but do nothing.

Other problems seem to be due to TEPaintText and the LaserWriter NT printing process. They are not reserved to DLT, I met them on all programs... when they use TEPaintText. When printing on US paper, it works fine but, with A4 paper (the one generally used in France) the first char of each line is partially erased (sample enclosed). It seems that this is an Apple problem. Enclosed is a piece of code allowing us to print complete lines. It may be of interest to you and your readers.

If one prints on a LaserWriter with TEPaintText (through DLT for instance) and launch Merlin 16+ he will be unable to reach the text Control Panel (tried with Merlin 4.0 to 4.8). With desktop programs I met no visible problem, but I feel that something is clobbered in memory and, [as a] matter of caution, I reboot when I have finished this kind of task. The problem has been encountered after each printing process on different machines, with or without TransWarp GS, with or without the Fast SCSI card, with Apple or non-Apple hard disks, with French or US GS/OS. ... The anomaly may be present in all cases but doesn't appear as it does when staying in desktop programs.

Are you aware of some quirks of the Fast SCSI card? If you pass under ProDOS8 and try to set prefix to the second partition of a hard disk, at the first try you will get a neat pause which seems a bit surprising. If you repeat the prefix change without quitting the P8 world, the infamous pause will not reappear, but, of course, if you reenter GS/OS then come back to ProDOS8, the same process will [happen]. It seems that some unknown scanning or building process is done.

If you try to boot from a ProDOS8 disk with Apple Talk selected (the standard state when owning a LaserWriter) and your hard disk off, you will get a 35 second pause which may end with a pretty "RELOCATION ERROR". If the floppy is a GS/OS, P16, or a UniDOS one the pause doesn't appear [sic]. In this circumstance the good choice is disconnect AppleTalk and boot. For my own use, I put on a special floppy [sic] a special block00 which quickly installs the needed Bram config on a first boot (need 1 or 2 seconds) allowing me to reboot with a single keypress on the desired device. With this tip I spare the infamous pause and my nerves don't [sic] suffer.

Back to TextEdit. I have two programs using it and allowing cut/copy/paste. One was written by myself. When using them to cut in 6/7 pages documents, I discover that sometimes words are broken. When this arrives [sic] stop the cut process or you'll hang the system. Put the cursor just before the second part of the broken word, read the last char of the first part, press delete once then retype the last char of the 1st part which may have been destroyed. All will be reset OK...

```

PushDLong mac
    pei    #1+2 ;of course I have to
    pei    #1  ;use a direct pg addr
    <<<

    wordResult
    PushDLong PrtRecHndl ;hndle to print
record

    _prJobDialog
    pla

    bne :Printit ;true = ok, continue

Printit: _WaitCursor
    ldy #prInfoSub+6
    ldx #6
:1    lda [PtrRectPtr],y
    sta rectangle,x ;copy (rect) Rpage
    dey

```

```

dey
dex
dex
bpl :1
ldy #prInfoSub
lda [PtrRecPtr],y
cmp #3      ;is it a LaserWriter?
bne :2      ;no
lda rectangle+2
clc
adc #6      ;adjust 'left'
sta rectangle+2 ;to accomodate A4 Prob
... here normal code ...
:PrintPage      ;using our modified rect
    LongResult
    PushLong PrtDocPtr ;a ptr to graf port
to draw into
    PushLong startingLine ;line # to print
    PushPtr  rectangle  ;ptr to rect to
draw into
    PushWord #0      ;flags
    PushDLong editTxtCtrlHndl
    _TEPaintText
    PullLong startingLine

```

Yvan Koenig
Vallauris, France

Dear Yvan,

Merci! Thank you for the DLT bug report and the SCSI and TEPaintText info. I'll forward those on to Apple DTS in case they might be able to make some use of the information.

Your TEPaintText fix indirectly raises an important point: we American software developers tend to totally forget about the rest of the world. I can attest to the fact that Europe is one of the hottest markets for 8/16 and Ariel Publishing products in general.

Y'all paying attention? It looks like I'll be attending a conference in February in Munich, Germany courtesy of the European Consortium of International Schools (over 700 schools represented). As I understand it, these folks are in dire need of 8 bit Apple II products. If I could sell one product to every school just in the ECIS consortium, I'd be a very happy camper, indeed.

Just something to think about...

Guten tag, mein Freund.

== Ross ==



Applesoft Auto Wordwrap

by Jerry Kindall, Classic Apple Editor

I wrote this month's Universal Text Output routine over nine months ago, for a program I was upgrading for a mail-order bookstore. The program was an on-disk catalog of all their books and was written in Applesoft. One modification that was requested was to compile the program using the Beagle Compiler, so I had to keep that in mind as I developed the rest of the modifications. The other requirements were that the program run on a II+ (which meant either using no lowercase or converting lowercase output to uppercase), that word-wrap be implemented for the book names and descriptions, that the list of book names be scrollable in both directions, and that an inverse cursor be used to select options and book names.

I decided to solve these problems with assembly language. The routine is generic enough to be useful to most Applesoft programmers, so I'll share it with you in this article.

Installing The Routine

I had originally intended to put the routine in page 3, home of most wayward assembly language routines meant for use with Applesoft. Unfortunately, the program turned out to be too large to fit in that space. I decided, instead, to install the program at the beginning of Applesoft program space, and to adjust Applesoft's start-of-program pointer to avoid overwriting the routine with BASIC code.

To make room for the routine, then, you can do one of two things. First, you could include the following line at the beginning of your main program:

```
10 IF PEEK (104) <> 9 OR PEEK (103)
```

```
<> 60 THEN POKE 104,9: POKE 103,60:
POKE 2363,0: PRINT CHR$(4);"RUN
program.name"
```

If your main program is sizable, the time required to load it twice may become distracting. (The program is loaded once at Applesoft's normal address, at which time it notices that it's not where it needs to be, so it adjusts the Applesoft pointers and reloads itself.) If that's the case, consider using a separate startup program consisting of the following line:

```
10 POKE 104,9: POKE 103,60: POKE
2363,0: PRINT CHR$(4);"RUN
program.name"
```

Your user would then run this startup routine to start up the program, rather than running the main program directly. Load time would be reduced because the lengthy main program would be loaded only once.

Once you have adjusted Applesoft's pointers to reserve the memory needed by the Universal Text Output routine, you can simply BRUN it, like this:

```
20 PRINT CHR$(4);"BRUN TEXTOUT"
```

The routine loads and connects itself immediately.

Using the Word Wrap

Once TextOut is connected, word wrap happens automatically. Whenever you print a character at the right edge of the screen, TextOut backs up to the previous space and

moves the last word on the line to the next screen line. You may see a slight flicker at the right margin as you print text; this is normal, because the characters are actually printed before being erased and reprinted on the next line. (Set SPEED=50 or so to see this happening.)

look nice on newer Apples while still running on older ones.

"Once TextOut is connected, word wrap happens automatically."

So to word-wrap a bunch of text, all you need to do is print your strings, without intervening carriage returns. Assuming that the array A\$ contains N strings of text (with an unknown number of characters in each string -- up to 255 -- it doesn't matter), all you have to do to print the text with word wrap is this:

```
50 FOR I = 1 TO N: PRINT A$(I);
   :NEXT: PRINT
```

One thing to be aware of: when using the automatic word wrap, you cannot print anything in the rightmost column of the screen. Attempting to do so will cause the last word on the line to wrap! So if you need to print in column 40, be sure to deactivate TextOut first.

By the way, the word wrap (like all TextOut functions) respects the text window set with POKES to locations 32-35. A 40-byte area at the end of the keyboard buffer is used during the word wrap process.

NORMAL, INVERSE, and FLASH

NORMAL and INVERSE work just as you'd expect them to, except that INVERSE works with lowercase. (TextOut activates the alternate character set.) FLASH is not supported; it is essentially the same as INVERSE (except that the first character printed after a FLASH command may be garbled). The Apple does not support both inverse lower case and flashing characters on the same screen.

Case Conversion

If TextOut detects that it is running on an Apple II+, it assumes that lowercase characters are not available and converts all lowercase letters to uppercase. Thus, your programs can

Screen Scrolling

TextOut includes two easy commands for scrolling the text screen (actually, the current text window) up and down. PRINT CHR\$(1);

scrolls the screen up one line, leaving the cursor where it was. PRINT CHR\$(2); scrolls the screen down one line, leaving the cursor where it was. This makes scrolling lists of text a snap, and adds a capability to 40 column mode that is usually only available in 80-column mode.

Text Highlighting

TextOut makes it easy to highlight text on the screen. Usually, if you're doing a menu with a moving inverse bar, you need to store the strings in an array and use VTAB, HTAB, INVERSE, NORMAL, and PRINT to display and erase each item as necessary. TextOut has a faster way. Simply set INVERSE or NORMAL display mode, position the cursor, POKE the number of characters to highlight into location 0, and PRINT CHR\$(3);. This command works with text already on the screen -- no need to store it and re-print it. Also, it's much faster than the old way.

The highlighting will wrap at the end of a screen line, so you could highlight three lines of text with POKE 0,120: PRINT CHR\$(3);.

Deactivating and Reactivating

Use PRINT CHR\$(4);"PR#0" to deactivate TextOut. (Actually, a PR# command to any slot will deactivate it. PR#1 to perform a printout, for example, will leave TextOut disconnected.) Use PRINT CHR\$(4);"PR#A\$800" to reactivate it later (or in place of PR#0 after PR#1). While TextOut is disconnected, word wrap and case conversion don't work, and CHR\$(1) through CHR\$(3) do nothing. You'll definitely want to deactivate TextOut before your program ends.

Listing One: TextOut Source Code

```
1 *Text I/O Hndlr for 40 Col Displays
2 *by Jerry Kindall -- Sep 90 8/16
```

```

3
4 *This rtn performs the following functions:
5 *
6 *1)Provide 40 col txt output on Apple IIs,
7 *automatically convert lower case to upper
8 *case on Apple II+ and allowing lower case
9 *in both norm & inv modes on IIe/IIc/IIgs
10*2) Perform word wrap at rt edge of scrn
11*3) Provide way to hilight & unhilight text
12*on the screen;
13*4) Provide way to scroll screen up & down.
14*
15*The above work only on t40 col text screen.
16*Fns 1 & 2 are performed automatically by
17*this rtn once installed. Functions 3 and
18*4 are performed by printing control chars.
19*
20*Control-A [CHR$(1)]:Scrolls scrn up 1 line,
21*leaving the cursor exactly where it was.
22*Ctrl-B [CHR$(2)]: Scrolls scrn down 1 line,
23*leaving the cursor exactly where it was.
24*Ctrl-C [CHR$(3)]: Hilights or de-hilights a
25*portion of the text scrn from the current
26*cursor pos. To hilight, go into INVERSE
27*mode before issuing this cmd. To de-hilite,
28*go into NORMAL mode. Before issuing cmd,
29*use POKE 0,x to specify how many chars to
30*highlight or de-highlight, up to 255.
31*
32* To install rtn, simply BRUN it. It loads
33*at $800 & requires 307 bytes. You must
34*POKE 103,60: POKE 104,9: POKE 2363,0 to
make room
35* for this code below Applesoft program.
36*
37* You can't use flashing text while this rtn
38* installed, since it allows inv lowercase.
(On
39* II+, only uppercase is allowed.) Use only
40* INVERSE and NORMAL commands while rtn is
41* installed.
42*
43* Doing a PR#0 will disconnect rtn. Use
44* PR#A$800 (as in PRINT CHR$(4)"PR#A$800")
to re-
45*activate this routine. Be sure to discon-
nect
46* when program run is completed!
47
48     org     $800     ;312 ($138) bytes req'd
49
50 * Zero page variables
51
52 count     =     0 ;how many chars to invert
53 psave     =     1 ;proc status temp locn
54 asave     =     2 ;temp accumulator save
55 xsave     =     3 ;temporary x reg save
56 ysave     =     4 ;temporary y reg save
57 base      =     $28 ;current text line addr
58 left      =     32 ;left margin of txt wnd
59 width     =     33 ;width of text window
60 top       =     34 ;top margin of text wnd
61 bottom    =     35;bottom margin of txt wnd
62 ch        =     36;cursor horizontal loc
63 base2     =     $2A;used during scrolling
64 invflg    =     50 ;$FF = norm, $3F = inv
65 prompt    =     51 ;0 if running program
66 csw       =     54 ;Monitor output vector
67 ormsk     =     $F3;FLASH mode: 0=off
$40=on
68
69 * Monitor entry points
70
71 bascalc   =     $FBC1 ;calc txt base addr
72 scroll     =     $FC70 ;scrll scrn up 1 line
73 idbyte    =     $FBB3 ;equals $EA if on II+
74           =     ;$06 for later models
75 cout1     =     $FDF0
76 altchar   =     49167;alternate char set ON
77
78 buffer    =     $2D8;40-char buffr for wrap
79
80 * Entry point - setup and initialization
81
82 entry     ldx     #nucout ;hook up nucout to
output
83           stx     csw ;while modifying code
84           stx     entry+2 ;at entry to read:
85           lda     #/nucout ; cld
86           sta     csw+1 ; jmp nucout
87           sta     entry+3 ;so that PR#A$800
works
88           lda     #$D8
89           sta     entry
90           lda     #$4C
91           sta     entry+1
92           rts
93
94 * Our new output handler
95
96 nucout    cld
97           php
98           sta     asave;save off all our regs
99           stx     xsave ;cause we're
gonna use 'em
100          sty     ysave
101          pla
102          sta     psave
103
104          ldy     prompt
105          beq     :cont
106          lda     asave
107          jmp     gocout
108
109 :cont     lda     asave
110          sta     altchar ;switch to alt
char set
111
112
113          and     #$7F;strip hi bit from char

```



```

114      beq   gocout      ;it's a null
115      cmp   #4         ;is it one of ours?
116      blt   goctrl     ;go process it
117      cmp   #33        ;compare to space
118      blt   gocout     ;it's spc or ctrl,
print
119      ldy   ch         ;are we at right margin?
120      iny
121      cpy   width      ;width
122      bne   gocout     ;nope, don't wrap it
123
124 wrap  jsr   docout     ;print it on screen
125 :backlp lda  #$88       ;backspace once
126      jsr   cout1
127      ldy   ch         ;get horiz cursor pos
128      beq   nowrap     ;we're at start,
no wrap
129      lda   (base),y   ;get character
at cursor
130      and   #$7F       ;strip high bit
131      cmp   #$20       ;is it a space?
132      bne   :backlp    ;nope, back
another space
133
134      ldx   #0         ;init index to buffer
135 :wraplp iny        ;bump up screen index
136      cpy   width      ;are we past width?
137      bge   :saved     ;yes, done saving
138      lda   (base),y
139      sta   buffer,x
140      inx
141      bne   :wraplp     ;always taken
142
143 :saved stx   count     ;remember count
144
145      lda   #$A0       ;space
146 :clearlp jsr  cout1     ;print it
147      ldy   ch         ;got to next line yet?
148      bne   :clearlp   ;nope
149
150 :bufout ldy   invflg    ;save old inv flag
151      lda   #$FF       ;set norm (verbatim)
152      sta   invflg
153      ldx   #0         ;pt to strt of buff
154 :outlp  lda   buffer,x  ;get a char
155      jsr   cout1      ;print it
156      inx           ;bump to next char
157      cpx   count      ;are we done?
158      blt   :outlp     ;no, do next
159      sty   invflg;restore inverse flag
160      jmp   exit      ;and go back to caller
161
162 goctrl blt   ctrl     ;always (get here with
blt)
163
164 nowrap lda  #$8D     ;load CR and fall thru
165
166 * Call docout and return to caller
167
168 gocout jsr   docout
169 exit   lda   psave
170      pha
171 eq     lda   asave    ;restore registers
172      ldx   xsave
173      ldy   ysave
174      plp
175      rts
176
177 * Call COUT1 with special handling
178
179 docout ldx   invflg
180      ora   #$80       ;restore high
bit
181      cmp   #$A0       ;less than spc (ctrl)?
182      blt   :1         ;print it always
183      bit   idbyte     ;do we have a II+?
184      bpl   :0;nope, check inverse mode
185      cmp   #$E0       ;is it lower case?
186      blt   :0         ;nope, check inv mode
187      sbc   #32        ;subt 32, convt to uppr
188 :0    cpx   #$FF;are we in normal mode?
189      beq   :1         ;yes
190      ldx   #$3F;force "true" inv mode
191      ldy   #$FF;force to $FF for now
192      sty   invflg
193      iny           ;force FLASH mode off
194      sty   ormsk ;(helps not for this
char)
195      and   #$7F
196      cmp   #$60       ;lower case?
197      bge   :1         ;yep, OK, print
198      cmp   #$40       ;symbol or ctrl char?
199      blt   :1         ;yep, OK, print
200      sbc   #64        ;bump ASCII code down
201 :1    jsr   cout1     ;print it
202      stx   invflg ;restore old inv flg
203      rts
204
205 * Handle our control characters
206
207 ctrl  cmp   #2;how does it relate to 2?
208      beq   scrldn     ;equal, scroll down
209      bge   hilite     ;greater (3),
hilight
210      jsr   scroll ;less (1), scroll up
211      jmp   exit
212
213 * Highlight (or un-highlight) characters
at cursor
214
215 hilite ldy   ch
216      lda   (base),y ;get char at
cursor
217      and   #$7F
218      cmp   #$2       ;is it an inverse
letter?
219      bge   :0 ;nope, go ahead print
220      adc   #$40 ;adjust to proper val
221 :0    jsr   docout ;print inv or norm
222      dec   count     ;are we done?

```

```

223         bne  hilite      ;nope
224         jmp  exit        ;yep
225
226 * Scroll screen down without moving cursor
227
228 scrlndn lda  base        ;remember
current base addr
229         pha
230         lda  base+1
231         pha
232         ldy  bottom      ;get address of
bottom line
233         dey
234         tya
235         sta  count
236         jsr  bascalc
237 :vloop  lda  base        ;sav old base
addr in base2
238         sta  base2
239         lda  base+1
240         sta  base2+1
241         dec  count        ;move up one line
242         lda  count        ;get line number
243         bmi  :clear;if neg, clear top
line
244         cmp  top          ;above top of window?
245         blt  :clear ;yes, go clear top
line
246         jsr  bascalc
247         ldx  width        ;how many chars
to copy
248         ldy  left         ;where we're starting
249 :hloop  lda  (base),y ;copy char from 1
line
250         sta  (base2),y ;to line below
251         iny                ;pt to next char
252         dex                ;count how many we've done
253         bne  :hloop        ;nope, next char
254         beq  :vloop        ;yes, next line
255
256 :clear  ldy  left         ;clear top line of
display
257         ldx  width
258         lda  #$A          ;normal space char
259 :cloop  sta  (base),y
260         iny                ;adjust pointer
261         dex                ;adjust counter
262         bne  :cloop        ;nope
263
264         pla                ;remember old base addr
265         sta  base+1
266         pla
267         sta  base
268         jmp  exit        ;back to caller
269
270         lst  off

```

Listing Two: TextOut Object Code

If you don't have an assembler, enter the lines below exactly as shown starting at an Applesoft prompt. Be sure to double-check your typing; you are entering important machine-language code!

POKE 104,9: POKE 103,60: POKE 2363,0: NEW
CALL-151

```

800: A2 19 86 36 8E 02 08 A9
808: 08 85 37 8D 03 08 A9 D8
810: 8D 00 08 A9 4C 8D 01 08
818: 60 D8 08 85 02 86 03 84
820: 04 68 85 01 A4 33 F0 05
828: A5 02 4C 8F 08 A5 02 8D
830: 0F C0 29 7F F0 59 C9 04
838: 90 51 C9 21 90 51 A4 24
840: C8 C4 21 D0 4A 20 9D 08
848: A9 88 20 F0 FD A4 24 F0
850: 3C B1 28 29 7F C9 20 D0
858: EF A2 00 C8 C4 21 B0 08
860: B1 28 9D D8 02 E8 D0 F3
868: 86 00 A9 A0 20 F0 FD A4
870: 24 D0 F9 A4 32 A9 FF 85
878: 32 A2 00 BD D8 02 20 F0
880: FD E8 E4 00 90 F5 84 32
888: 4C 92 08 90 42 A9 8D 20
890: 9D 08 A5 01 48 A5 02 A6
898: 03 A4 04 28 60 A6 32 09
8A8: 10 06 C9 E0 90 02 E9 20
8B0: E0 FF F0 15 A2 3F A0 FF
8B8: 84 32 C8 84 F3 29 7F C9
8C0: 60 B0 06 C9 40 90 02 E9
8C8: 40 20 F0 FD 86 32 60 C9
8D0: 02 F0 1E B0 06 20 70 FC
8D8: 4C 92 08 A4 24 B1 28 29
8E0: 7F C9 20 B0 02 69 40 20
8E8: 9D 08 C6 00 D0 ED 4C 92
8F0: 08 A5 28 48 A5 29 48 A4
8F8: 23 88 98 85 00 20 C1 FB
8A0: 80 C9 A0 90 24 2C B3 FB
900: A5 28 85 2A A5 29 85 2B
908: C6 00 A5 00 30 15 C5 22
910: 90 11 20 C1 FB A6 21 A4
918: 20 B1 28 91 2A C8 CA D0
920: F8 F0 DD A4 20 A6 21 A9
928: A0 91 28 C8 CA D0 FA 68
930: 85 29 68 85 28 4C 92 08

```

3D0G

BSAVE TEXTOUT,A\$800,L\$138

Insecticide

David Gauger's Hardware Hacker column in the July issue (the II-Ears voice recognition project) contained a buggy (?) circuit diagram for the DB-9 version. Here's the corrected schematic. David sends his apologies and hopes that no one was inconvenienced too much.

Schematic Diagram - DB-9

Use with Apple IIe, IIc, IIc+, IIgs

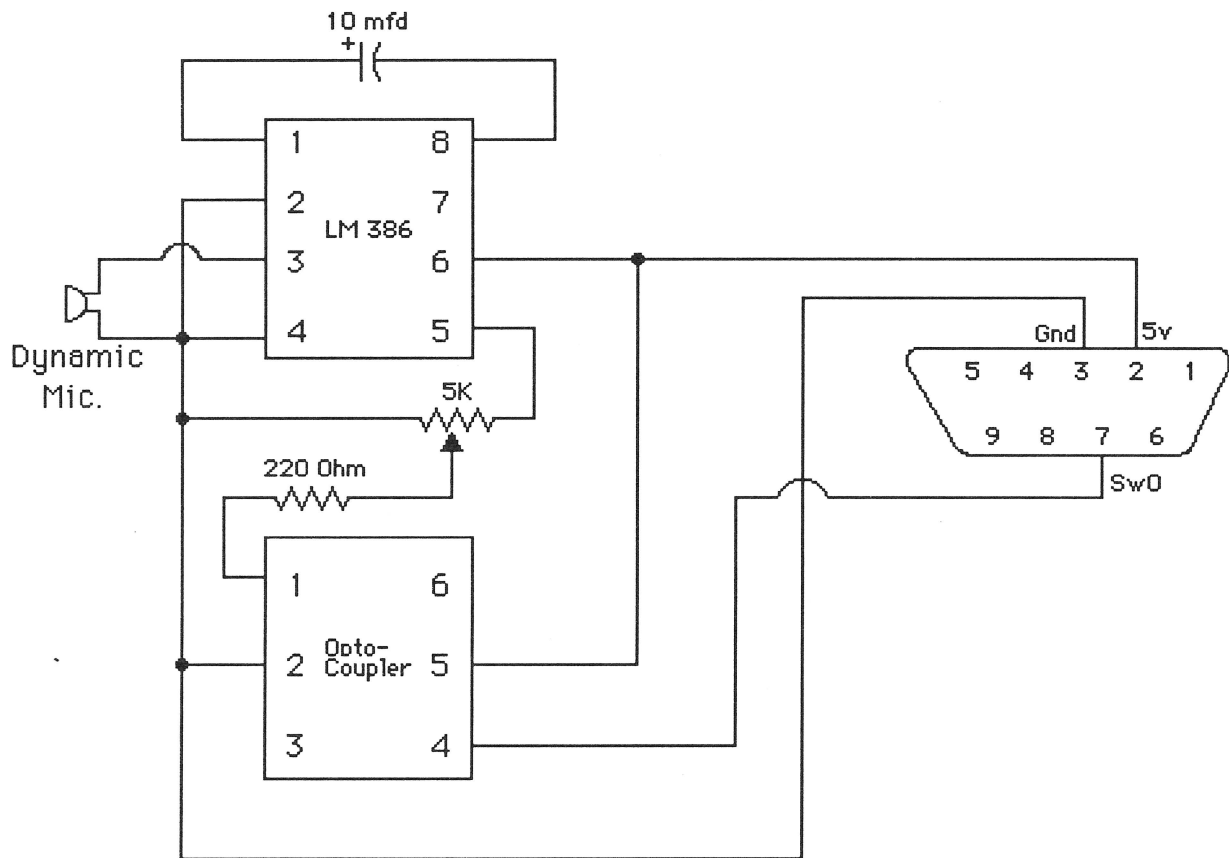


Figure #2

II-Ears Speech Recognizer

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!):

- **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.
- **May '90:** 8 bit - Tom Hoover's AppleWorks Style Line Input.
16 bit - Bryan Pietrzak's shell utilities for Orca/APW, Steve Lepisto's Illusions of Motion, Number Two.

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 concerned with the content of your interactive fiction program as with the form. This package is fun, entertaining, and useful. It includes Applesoft, ZBasic, and Microl 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 Microl versions is that with the easy integration of text and graphics provided in those languages, 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.

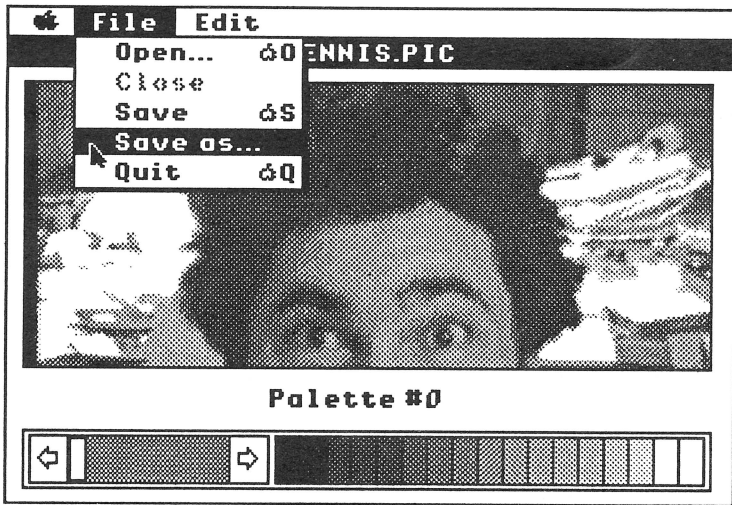
All back issues are \$3.00 each (postage and handling included except for non-North American orders. Those of you on other shores please add \$1.50 extra per issue).

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.



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

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.

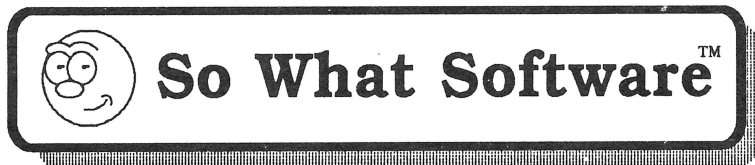
The Call Box TPS \$99.00

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 to 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.....	36,37
Bringardner Data Products	15
Direct Micro.....	40
Kitchen Sink Software.....	11
Ron Lichty	22
LRO Computer Sales.....	16
Night Owl Software	2
So What Software.....	38
SSSi.....	23

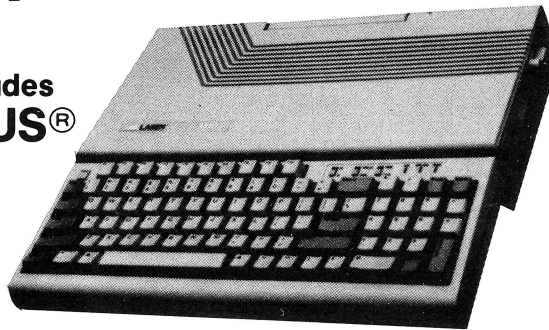
BULK RATE
U.S. POSTAGE
PAID
PATEROS, WA
PERMIT NO. 7

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

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

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

Your satisfaction is our guarantee!

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.

<http://apple2scans.net>