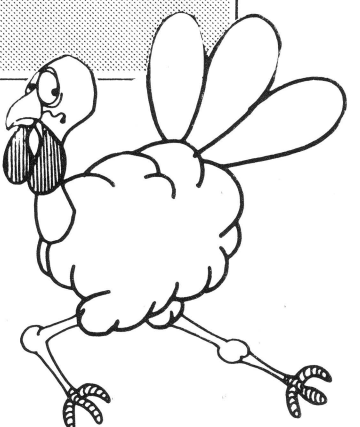
November, 1990
Volume 1, No. 9

8/16

The Journal of Apple II Programming $4.00

Picture Perfect in Pascal: Phil Doto Decodes SHR

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

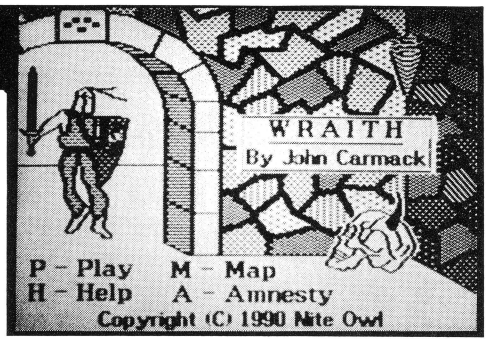


New

WRAITH

Adventure

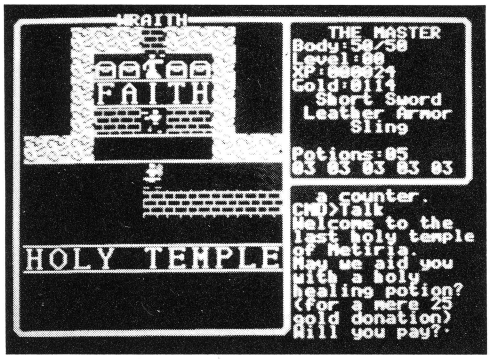
Special Introductory Price $9.95



The **Devil's Demise** - is an exceptional graphic adventure game that comes complete on a single 3.5 inch disk with on-screen instructions, a map, demo play option, and dungeons which were too vast and expansive to fit on 5.25" disks.

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

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



"I have never in my life seen a better way to spend such a modest amount of money", **writes Neil Shapiro in his review of WRAITH in the July 1990 issue of Nibble magazine.**

√√√ *Font Collection* √√√

The A2-Central staff has spent years searching out and compiling hundreds of IIGS fonts. These fonts are packed onto eight 3.5 inch disks. They work with IIGS paint, draw, and word processing programs. Includes a program to unpack them, an Appleworks data file that lists the available fonts, and picture files that let you view the various fonts.

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

In Depth:

Close Out!

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

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

Keep it Cool and Quiet

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

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

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

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

Use this handy Cut & Paste address label for fast service

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

• **School Purchase Orders are Welcome** •

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

Ship to:

Telephone #:

Credit Card or PO# | Expires

Quantity	Description	Price	Amount
	Slide-On Battery Kits	$ 14.95	
	Nite Owl Journal 3.5	$ 9.95	
	WRAITH Adventure	$ 9.95	
	GS Super Cooler Fan	$ 24.00	
	Font Collection	$ 39.00	

Signature for Credit Card Orders

Kansas Sales Tax

Shipping & Handling

TOTAL

Please include $2 shipping and handling / $5 for overseas orders. Kansas residents add 6% sales tax.

Prices may Change Without Notice

8/16

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

Subscription prices in US dollars:

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

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

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

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

News, Blues, and Marketing 102

Roger Wagner Publishing announced both HyperStudio version 2.1 and the HyperStudio runtime module. Version 2.1 has numerous improvments in both paint tools and stack editing (you can insert a card in the middle of a stack now, for example). Best of all, the new update is *free* to registered owners who request it. Folks, ya gotta love (and reward) this kind of loyalty to customers.

The runtime module is a pretty big deal for those who have commercial quality stacks that they want to sell (there is a nominal license fee). Your market now extends to *all* GS owners, not just HyperStudio owners. This is the first time that I have personally been tempted to enter the stack creation game at a serious level. Roger's number is (619) 442-0522. You and I both should probably give him a call.

In another "customer first" gesture, RWP has released a very special pair of HyperStudio XCMDs to the online services (or you can purchase them directly from RWP for a mere $10). These two add-ons make stacks more accessible to those folks using a switch controlled input device (such as those folks with physical disabilities might use - for more info about switches, see David Gauger's Hardware Hacker column this month. Perhaps you can design both hardware and software for the handicapped!). One XCMD highlights all of the buttons on a card one by one. If you toggle a switch while a button is highlighted, it is selected. Cool idea. The other scrolls text automatically (I don't know the details on this one).

My hat is off to Roger and Co. for their attention to this oft neglected market, and for making the software easy to acquire.

Since my wife is a special education teacher, I'm pretty familiar with the difficulties the handicapped sometimes have with off the shelf software. I can tell you straight out that the special education marekt is easy to identify and inexpensive to market to (via mailing lists and educational journals). There is money to be made here. Get to it. You can have the dual pleasures of making a buck and really helping someone. Good capitalists are not necessarily selfish! (oops, a little of my politics creeping in there...)

Complete Technologies, Inc. recently acquired all of the IIgs languages produced by the TML folks. The head honcho at Complete is none other than Vince Cooper, the former IIgs product manager/developer for TML. This bodes well for TML Pascal and TML Basic owners. I think that Vince will be giving you (and your computer) the attention you deserve. Congratulations, Vince.

The following is yet another tirade of epic proportions. Read at your own risk.

Before I get too wound up, let me emphasize that I am fully aware that the October 15th Mac rollout was never billed as anything but a Mac rollout. I was not the least surprised that the Apple II was totally ignored, nor was I upset by it on its own merits.

What upsets me is that the Apple II has not had a significant rollout of its own since the 1986 introduction of the computer. Yeah, the SCSI card is nice, as is the graphic overlay card. But where would the Mac market be if the SE and a couple cards had been the only additional product in the same time span? The cards and the extra RAM are bread crumbs. Call me an ingrate if you want, but they are just bread crumbs to me.

With that excuse, I now unsheath the sword and begin slashing...

Apple Computer, Inc.'s October 15th rollout of the new Macintosh models has many IIgs developers singing the blues. It has me again shaking my head in bemusement about Apple's marketing plans. I'm rooting for the success of the new Mac's (I'm a Mac developer, too), but Apple's insistence that the "Macintosh is the best buy for education" (almost a direct quote from one of the presentations in a major metropolitan area), has me in near hysteria (from both laughter and fear).

Somebody send me the name of an Apple marketing policy maker and I'll send a copy of this issue with the following highlighted in magic marker:

An open letter to Apple's marketing policy makers:

Dear Apple Marketing,

READ MY LIPS: You will not "drive" the education market in the same fashion that you've attempted to direct the business and consumer markets (and since when is a fairly discouraging 10% of the market really considered "driving" anybody, anyway?) Educators have their own agenda, and Apple marketing does not seem to have the slightest clue as to what it is.

I ought to know, I have only recently left the teaching profession. My wife still teaches.

The Macintosh does have its place in education, perhaps in the offices and journalism departments. But when the cost of software is factored in, the *Apple IIgs is the best buy for education.* Do you know what the one single lesson educators learned from the "computer revolution" of the 1980's was? A computer ain't worth (fill in your favorite expletive) without software. It took teachers 10 years to evaluate and acquire the software they now have. Apple is asking educator's to throw away their entire software library and buy a Mac.

We live in fear of doing that. We won't do that. The Apple II emulator card is not an answer, either, as many schools have invested heavily in GS software. Besides, if you abandoned us once, how do we know you won't be saying the same thing about some future Apple in a couple years? ("Yessiree, ladies and gents, get this neat new totally incompatible machine we call the Golden and you can get a Mac emulator for it. The Golden is undoubtedly the best buy for education....")

It doesn't take a rocket scientist to recognize the ol' bait and switch. And that is what current Apple marketing strategy feels like to educators. Contrast that to IBM software. The vast majority of 'wares developed for the original PC will run just fine on a screamin' state of the art 486 (yukky though they be). *That* is the real reason educators are switching to MS-DOS. They sense a stability and a continuity.

"I guess that if a 10% market share is considered successful in the business market, maybe it is okay for education, too, eh?"

Schools are willing to buy IIgs's in the same manner they acquire all new equipment - gradually, as older things wear out. That the schools didn't run out and buy IIgs's in 1988 as fast as they bought II's in 1983 should be no surprise (espcially in light of the goings-on at Apple at that time). To think educators will spend money otherwise is to misunderstand the education market entirely. Which is, of course, what Apple has done.

There's one other issue totally missed here, too: educators are only part-time computer users. Very part time. The thought of having to learn a new machine - no matter how "friendly" - petrifies the bulk of the lot. That probably sounds bizarre to you all, but I *know* that it is a sort of subliminal factor in educator's somewhat knee-jerk reaction to the Mac.

Patience, persistence, and some *attention to the II line (i.e. a new GS CPU with real marketing support)* would go a long way towards getting educators to buy IIgs's. Let me emphasize the point: the IIgs must have some marketing - on television as well as in print - if Apple has any real hope of making money in education. If Apple reps hocked GS's to educators as hard as they're pushing Macs, they'd find the GS much easier to sell than the Mac. Educators want to buy GS's, but Apple seems to be looking the other way and offering excuses when asked about it. "We cannot comment on unannounced products. But we got some terrific Macs coming out..."

Am I the only one that sees the incongruity there?

I guess that if a 10% market share is considered successful in the business market, maybe it is okay for education, too, eh?

Oh, but there's one last gotcha, Mr. Apple marketer: you may want to bleed the last dollar from the II market with no expenditure of marketing or hardware, but there aren't going to be any developers left writing for your bleeding machine. Already many - if not most - of the best have left for Amiga and MS-DOS land. A few have gone Mac (like me), but it ain't the majority by any means.

My Plan

I'll quit throwing stones and outline a promotion that would make the II a billion dollar machine again (Alright, I'm conceited and I admit it. I really believe that I could sell a jillion GS's). By the way, it is not really the details of my ideas that are important, anyway.

The theme is "Home Work". The entire campaign would focus around two images. The first is Joe Jr. He's working on a IIc+ at school, maybe doing something in AppleWorks. Then he comes home and finishes up his project on a IIgs (or better yet, an 8mhz IIgs+).

The second image is that of Joe's father (or mother), slaving away over a spreadsheet at the office on an obviously blue machine (if that is offensive, then make it a Mac). When Joe Sr. comes home, he pops his disk into his GS and starts working on the same spreadsheet, easily imported into AppleWorks GS.

The closing line: "The Apple IIgs: for your Home Work." If it is a TV spot, the announcer ought to pause slightly between "Home" and "Work".

These images and that simple statement would convey a wealth of information. First, the IIgs is compatible with all of Joe Jr.'s software at school. A large percentage of home computers (I believe) are sold with children in mind. This push for the home market would pay off with increased sales in the education market. We teachers *want* our students using computers at home because we *cannot* give them enough time on them at school. It is nigh unto impossible. **The home market and the education market play off each other.**

The point of seeing Joe Sr. popping his disk into the IIgs and working on his spreadsheet (this necessitates MS-DOS and Macintosh FST's!) communicates the notion that real work can be done on the GS, and that it can read the file formats of other computers. We all know this is no stretch of the truth! And if Apple would bundle AppleWorks GS with the thing...

I bounced my ideas off a friend, and he replied that the only drawback was that Apple might perceive it as a threat to Macintosh sales, and they'd never go with anything that might hurt the Mac at all.

I think the stockholders want to go with what will make the most money. What is more profitable, the sale of one Mac or five IIgs's? I believe the IIgs could outsell the Mac in the home and education markets to that degree *if properly marketed.*

The education market is, by itself, a mighty force in the computer market. And it is my proposition that it can and will influence the home market, provided that the target computer is also portrayed as compatible with the business world.

The time for a media campaign such as this is now. Not only is the II market in dire need (unlike ever before, I think), but the personal computer market in general is also starting to accept "stratification".

By stratification I mean that consumers are becoming reasonably aware of the fact that the latest and greatest is not necessarily for them. IBM is betting literally millions on this with the weenie PS/1. Furthermore, one need only look at the automobile market to see a parallel: very few buy state of the art cars. What we're after is value for the money.

If Apple plays up the desktop interface on the GS (some Mac and DOS people I know were not even aware that the GS had such a thing - or even a toolbox), and if it provides a 20 meg hard drive, and especially if it comes out with a 2 meg (in RAM), 8 mhz machine, you could charge $250 more than the comparable PS/1 and still compete with it (more RAM, better interface, more sophisticated operating system, etc.)

But that's a lot of "and ifs".

Finally, Mr. Apple Marketer, there is the matter of trust and integrity, two very sensitive areas I've never broached in print before. Desperate men do desperate deeds...

As a developer, I believed you when you said "Apple II Forever". I believed you when put the words, "The Apple II is still a very important part of our business..." into John Sculley's mouth (or words to that effect).

If Apple goes back on its collective word, you'll have turned your most loyal customers - you're best sales people, doggone it - into enemies. You'll have turned a 70+% market share in education in 1986 to the more Mac-like 10% by 1996.

Do you want to know what IBM thinks? Some of their marketing reps in the field were reported by my sources to have said something to this effect: "Were very happy with the new Macs. We're just glad Apple didn't start marketing the GS. That would scare us."

You have already blown a lot of opportunities, I think. The question in my mind is how much repair can you do? I really hope that the "unannounced products" that keep getting tossed around are meat and not more bread crumbs. But all the fancy hardware in the world is worth nothing unless the product is marketed.

Hence I address this plea to you, Mr. Marketing policy maker, whomever you are. Make me eat crow. Make me swoon with delight over the new level of interest Apple has in my machine. Do it fast, or I'll have to start fighting you instead of supporting you.

Sincerely,

Ross W. Lambert, President
Ariel Publishing, Inc.

Marketing 102: Marketing for Small Developers Who Want to Survive to be Big Developers

Class will be considerably shorter than normal today due to the fact that our fair professor spent far too much time blowing off steam. He feels a whole lot better, however.

Today's class is entitled "Pre-development Planning". Yes, that's right. I said "pre-development".

Most small software companies (and their owners) tend to develop a product first and then try and figure out how to sell it. "Hey, I got this really hot algorithm for calculating pi to more decimal places than has ever been achieved on a personal computer...." The code and software may indeed be elegant and a work of art, but this is pretty much a backwards way to develop a product.

It is not an impossible situation, mind you, but it is not the best situation for making money. And let's be frank - if you are not profitable, you will not survive to write more great code.

For small companies with miniscule budgets (under $20,000 for marketing), the best market to attack is a highly specific, easy to find group with some intense need. My best advice is to get a catalog of mailing lists and look at the different groups represented. You may be surprised at the vast variety of lists you can rent. If you had a program that tracked family lines for dog breeders, etc., you could undoubtedly find a list broker who would rent you a list of such folks. Better yet (and for an additional fee), the broker would be able to screen the list for computer using dog breeders.

Hard to believe, perhaps, but very possible. Here's a tiny sampling of the lists available from one source (Research Products Group):

- Accountants
- Appliance stores
- Bicycle dealers
- Carpet installers
- Rabbis
- Aircraft dealers
- Babies (newborn) (!!!)
- Candy manufacturers
- Churches (by denomination)
- Saddlery & Harness shops

... and there are literally thousands more. If you don't get an idea for some software you could write looking through these lists, you probably ought to give it up right now.

If looking at people by profession or association is not enough for you, there are also thousands of publications who rent their mailing lists. For example, you could do a mailing to the readers of:

- Apartment Management Newsletter
- Bridal Club of America
- DVM Magazine (vetrinarians)
- Environmental and Waste Management World
- Business Week
- nibble
- inCider
- Byte
- Dr. Dobbs Journal

... and so on.

I've included some key addresses and contacts in the box at the end of this article. These folks will send you their catalogs free of charge. Call them.

This brings up a key point: if you have a small amount of marketing dollars, don't try the shotgun approach. Make certain every advertisement you pay to have printed gets into the hands of someone who is very likely to be a potential buyer of your product.

For folks marketing their own 'wares, this probably means some kind of direct mail campaign. It also means that you decide what software to develop based on your ability to reach a given market. Surprisingly enough, the general consumer market is the hardest to reach and the most expensive.

If you create a product that some small demographic slice of America absolutely has to have, they'll come looking for you. If you create a product that everybody has to have, you'll have to pay mass bucks to go looking for them.

It is one of the great paradoxes of our time, but it is quite true.

In spite of my admonitions, the question that many of you are going to have is what to do if you've already got a product and you need to find a market for it.

The cheapest means to your end is to go ahead and get all of the mailing list catalogs I've mentioned. Search through them and constantly ask yourself, "Do these people have a need for my software?"

For any group that you can answer yes about, highlight their entry in the catalog. Then ask yourself how many in that group should have a strong desire to purchase your product. Write down a percentage. Go with your first guess - don't think about it too long or your own optimism will start to influence you.

When you're done, select the top two or three groups and try test mailings to them. We'll get to testing and ad creation in a future installment of this column. Let me say right here and now, however, that if you want to survive to make any money you have to listen to what the market is saying. If your test mailings to your best possible target groups have abysmal returns, cut your losses and move on to a different product.

This is very hard to do. I've seen folks throw a whole lot of good money after bad trying to sell a product the market didn't want. I'll provide more details later, but you can tell if your product has a chance with marketing expenditures of under $3000. You most definitely do *not* need a $100,000 loan from a venture capitalist. That's a great way to ruin your life.

A slightly more expensive route is to have a mailing list broker do the research for you. Many of them will listen to your description of your product and

then try to dig up lists that might be appropriate. Debbie Stanley at InfoMat has done this for us. Her address is at the end of this article.

Collaborate!

Back at the pre-development ranch, you ought not let your own inexperience in a field cause you to toally rule out writing software for it. The best software for a given market is usually designed by a member of that market, anyway. You therefore undoubtedly need a collaborator (unless you're a very experienced member of the group you're targeting). If you're writing something for high school principals, your best collaborator is a high school principal.

It is also very important to do some research. Even if you think you know a field well, it is often instructive to talk to several others in the field. The program you'd like to have in your office may need a few more features or more flexibility in order to be the slightest bit useful in my office. Talking to people (I believe some call this a market survey) can provide you with tips that will greatly improve your chances for success.

In short, never develop software in a vacume. Don't do it alone. I have collaborated with a lawyer, an accountant, an artist, another programmer, and another teacher (with varying degrees of success, but every project that I've finished has been profitable). I've written some mighty strange software - from databases that report on molecules to programs that record student answers from a scanner.

My point: you don't need to write AppleWorks to be a professional programmer. In fact, I'd bet that 99% of the working programmers today do not ever have their products advertised in a national magazine.

Some of the most lucrative projects you could undertake would involve reaching a small but desperate market. You wouldn't believe the garbage that people are paying big bucks for.

If you do a better job than the other guy - or reach a market first - the rewards can surprise you. You may even make more money selling your own 'wares this way than the world famous programmer who only got a 5% royalty.

I hope this foray into marketing is helping a few of you. It's not source code, I know, but I've had enough questions on the subject to lead me to believe that the information is useful and practical. And that is what we want *8/16* to be. = Ross =

Mailing List Brokers

InfoMat, Inc.
Debbie Stanley
1815 W. 213th St., Suite 210
Torrance, CA 90501
(213) 212-5944

• Debbie has done great work for us in digging up lists that meet our criteria.

Research Projects Corp.
Pomperaug Avenue
Woodbury, CT 06798
(800) 243-4360

• Research Projects has the best general purpose consumer list collection I've seen.

MAL DUNN/GSC
Marion L. O'Neill
7101 Wisconsin Ave. Suite 1001
Bethesda, MD 20814
(800) 873-5478

• Managers of many publications lists.

Worldata
500 N. Broadway
Jericho, NY 11753
(516) 931-2442

• These are the caretakers of Apple's own customer list. They have many other large corporate clients.

Semaphore Corporation
207 Granada Drive
Aptos, CA 95003
(408) 688-9200

• Semaphore has a great collection of computer lists.

Market Data Retrieval
400 Oyster Point Blvd, Suite 301
So. San Francisco, CA 94080
(415) 871-0936

• MDR is, without question, the only place to go for education lists.

GENESYS

GS Sauce RAM Card

Now available and shipping!

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

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

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

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

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

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

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

Retail Price: $150.00

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

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

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

Memory configurations:

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

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

Pricing:

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

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

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

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

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

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

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

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

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

MasterCard VISA AMERICAN EXPRESS

SSSi

PushPin Studios, St. Augustine, Fl.

Classic Apple Hacking

But did he eat crow?

GenDraw: Jay Does 8 bit

by Jay Jennings

(In spite of his protestations to the contrary - check out Jay's Hired Guns ad elsewhere this issue - Jay does indeed do 8 bit programming. This article is the first in a series on 8 bit animation techniques. Future topics include pre-shifted shapes (applicable to just about any animation job, GS, 8 bit or otherwise) and double high resolution techniques. Since this is the first of many articles, please feel free to send in questions for Jay. I'll forward them to him as fast as possible and he'll include the answers in future installments. Please remember that we have about a 7 week lead time for articles so you might not see a response for a little while. - Ross)

I've done several "generic" articles for Ariel Publishing over the last few years, but all of them have been Apple IIgs specific programs. Well, all of them until now. I had sworn off doing 8-bit programming until Softdisk waved some greenbacks in front of my face. I changed my tune quickly.

Unlike the IIgs with its extensive toolbox, you have to almost start from scratch when writing an 8-bit program. This led to the creation of GenDraw, a very simple Hi-Res shape drawing routine that allows you to place shapes (space ships, little doggies, or whatever your artistic brain can come up with) on the high resolution graphics screen of the Apple II.

Before we get in too deep, make note of the following. The shapes we're talking about in this article are bit-mapped shapes, not the kind of shapes that Applesoft uses. An Applesoft shape is made up of a set of vectors, or directions. When an Applesoft shape is drawn, the computer looks at the shape and says to itself, "Start drawing here, move up 4 pixels, to the left 10 pixels, down one pixel..." And it continues in that meandering fashion until the shape is drawn. Yeah, it's okay for Applesoft, but if you want to get into arcade quality animation, you'll need to switch to bit-mapped shapes. Basically, a bit-mapped shape is a sequence of bytes that can be stored directly to the Hi-Res screen. We just shove the bytes straight to the screen, and the shape appears.

I'm not going to get into how to draw a bit-mapped shape. Instead, I want you to bug Ross to run another article on how to do that. And by some odd coincidence, I just happen to have one of those very close to finished <grin>.

Using GenDraw

To use GenDraw all you have to do is push the address of the shape to draw on the stack, push the width and height of the shape, and then load the X and Y registers with the horizontal and vertical location where you want the shape drawn. Then do a JSR GENDRAW and your shape will be drawn.

The first thing the routine does is to save the values in the X and Y registers because it then uses those registers to hold the return address while the program pulls more parameters from the stack into storage locations. This works fine for our purposes, but all that pushing and pulling takes up quite a bit of time. If you need extra speed, or have several shapes on the screen at once, you need a faster drawing routine. The easiest thing to do is to get rid of the PHA/PLA parts and just store the needed values in the memory locations directly before calling the drawing routine. I used this method for convenience sake *(Ed - I'm glad he did, too, because it makes it easier to incorporate the routine into high-*

er level languages. - Ed).

The sample program that shows off GenDraw runs under BASIC (so that I didn't have to do any of those SYS file things) and simply turns on the Hi-Res screen, clears it to black, prints the word HELLO twice, and then runs a little box from the left side to the right. Typing BYE after it stops moving will return you to your program launcher. Or type TEXT to pop back into BASIC.

Bash-N-The-Code

Let's look at the first part of the program - that part that gets the screen ready for graphics use.

The first three instructions in our program turn on graphics mode, select the full screen (as opposed to a graphics screen with four lines of text at the bottom), and chooses Hi-Res graphics rather than Lo-Res. Then we call a subroutine that goes into a simple loop plotting a zero byte (the color black) in all the locations of screen memory. Oh, please say you know what I mean by screen memory! In case your brain is a little fuzzy, just remember that the display on the Apple II is mapped into memory. That means that if you put a value into the range of RAM that is screen RAM, it shows up on the screen. Of course, if you didn't already know that, this article is probably already over your head. Tell Ross you want a tutorial on 8-bit graphics. You know, I could probably whip one of those together, too. <grin>

As an example, type the following in from BASIC:

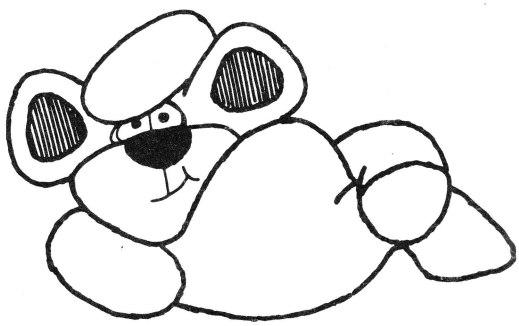
```
HGR
POKE 12288,255
```

A little white line should appear in the upper left hand corner of the graphics screen - five lines from the top. Voila! You just plotted a bit-mapped shape...well, kinda. The same principle is used in our routine.

Once the screen is turned on and cleared to black, we initialize a few variables and then use our handy dandy DrawShape macro to put the letters to the word HELLO on the screen. The DrawShape macro was created simply to make our source code more readable. If you take a look at it, you'll see that it just pushes the correct parameters and does a JSR GENDRAW.

After we print the words to the screen, we immediately go into our drawing loop. This loop draws the current shape, updates some variables, and then keeps looping. This continues until the shape we're drawing has gone the entire length of the screen. Then we bop out of the loop, and return to BASIC.

If you haven't done any Hi-Res animation before, you may be wondering why I have seven shape tables but only one shape is visible on screen. In reality, you're seeing all seven shapes. They're just all exact duplicates except that they're shifted over one pixel from the previous shape. Yikes! We're getting into pre-shifted shapes! Hmmm...I can see this article growing beyond the bounds of this magazine. Why do we need shifted shapes? Because when we draw it is far and away fastest to draw on byte boundaries (there's less calucation). And if we want smooth animation, we don't want to jump a byte at a time. So we make seven shapes, each shifted over one pixel from the previous, and then plot those shapes at the same screen byte location. If we plot all seven shapes one right after the other, it appears that the shape is moving to the right one pixel at a time. When we hit the last shape, we scoot to the next byte, and start plotting our seven shapes all over again.



Jay's Teddy Bear

Pre-shifted shapes can be confusing when you're just getting started in animation. Heck, everything about animation can be confusing! By using the GenDraw demo program as a convenient starting place you can play around with different sized shapes, change coordinates, etc. If you want to make a teddy bear travel across the screen instead of the little box, the first thing you need is a teddy bear shape. I've included one with this demo. Yes, the teddy bear looks more like a martian that's been stomped on, but trust me, it's a teddy bear.

You'll need to change the source code for this to work (do your experimenting on a copy of the program so you don't have to type it all in again in case of an accident). Cut out the seven shape table lines (Shape1 through Shape7) and type in the teddy bear shapes. Then change the two variables at the front of the source code to reflect the new size of the shape. Assemble it, run it, and watch that teddy bear move.

Math Anxiety

Let's take a look at an important part of the program. Right in the midst of the GenDraw subroutine we need to calculate where we're going to be drawing the shape. Since the graphics memory isn't contiguous[1], we have to do some fiddling to find out where in display memory we start drawing. First thing we do is to use the current vertical coordinate as an index into the two tables at the bottom of the source code listing.

Those two tables contain the starting addresses for all the lines on the Hi-Res screen. For example, if we want our shape to start on the fifth line, we can see that the fifth value in both tables gives us the starting address of that line, $3000 (or 12288 in decimal). Remember those two lines of BASIC code you tried a little bit ago? Proof that the lookup tables work! <grin> Thankfully the video memory is contiguous across each screen line, so all we have to do now is to add the horizontal coordinate to our line address, and we have the starting address for our shape.

This generic drawing routine works quite well as the core of a Hi-Res character generator. That's a program that allows you to print text on the graphics screen, much like we printed the words HELLO. In fact, I created some routines and macros that allow you to write assembly language code like the following:

```
Locate #10;#10        ;position the cursor
Print  "This is text on the graphics screen!"
```

Those two lines of code will position the "cursor" on the Hi-Res graphics screen where an HTAB 10 : VTAB 10 would have placed them on the 40 column text screen, and then the second line draws the given line of text to the graphics screen. Routines like that make creating programs much easier. I'm a big believer in "working lazy." I only had to develop the routines once and now I can use them in all future 8-bit programs. Some of these routines will appear in future issues (if there's any demand for them).

Also, with the addition of a few lines of code we can make our shape drawing routine work with double Hi-Res shapes. Because of the idiosyncrasies of double Hi-Res, you can't have a very efficient generic drawing routine, which is why I didn't implement it right now. But we'll look at one way to do it in an upcoming article. Also, we could look into creating a nifty shape drawing utility. The one I use was written by some novice programmer <ahem> about four years ago. It was written in Applesoft so it's slow. And the programmer <ahem> didn't know what he was doing at the time so it's kinda lame. It's called ShapeMaster and it should be on the 8/16 disk this month. You can also get it from the A2PRO Library on GEnie.

What I need from you is some feedback. Ross said we could start an 8-bit graphics column. But if nobody cares about it it'd be wasted space. So let me know (via Ariel) what kind of graphics stuff you'd like to see. Tutorials starting at the very beginning? More advanced techniques? A mix of the two? - Jay

1. Woz interleaved the screen RAM in a near-spaghetti fashion for a purpose - it cut down on the electronic requirements of the video circuitry. In short, it made the original Apple's faster and cheaper. 'Tis just another example of his genius at work, really. - Ross

Listing 1 - GenDraw Macros

(note: all listings were created with Merlin 8/16)

```
DrawShape mac
      PushWord ]1     ;shape address
      PushByte ]2     ;height of the shape
      PushByte ]3     ;byte width of shape
      ldx   ]4        ;X coordinate
      ldy   ]5        ;Y coordinate
      jsr   GenDraw   ;go draw the shape
      eom

PushWord MAC
      IF    #=]1
      lda   >]1
      pha
      lda   ]1     ;lo byte
      pha
      ELSE
      LDA   ]1+1
      PHA
      LDA   ]1
      PHA
      FIN
      <<<

PullWord mac
      pla
      sta   ]1     ;get the lo byte
      pla
      sta   ]1+1   ;get the hi byte
      eom

PullByte mac
      pla
      sta   ]1
```

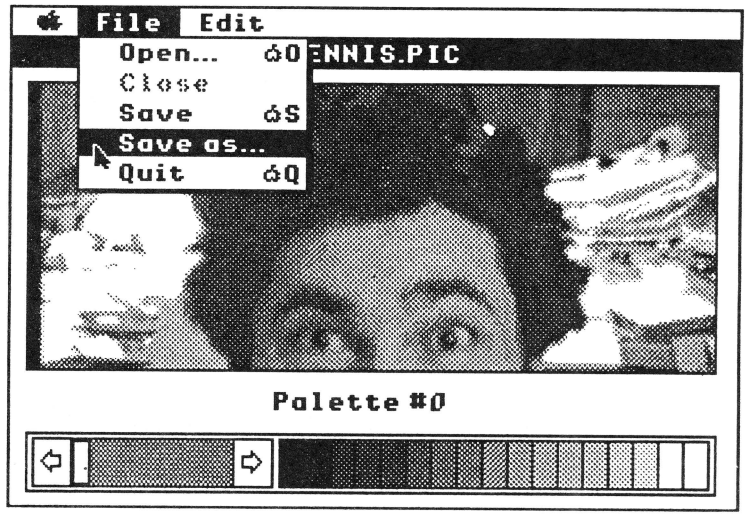
```
eom

PushByte mac
      lda   ]1
      pha
      eom
```

Listing 2 - GenDraw Demo

```
      lst   off
*=======================================
* a little demo to show our Generic
* Drawing routine
* Another Mohawk Man Creation
* Copyright 1990 - PunkWare
*=======================================
      xc          ;allow 65C02 opcodes
      mx    %11   ;only 8-bit, thank you.
      cas   in ;this isn't C (thank God)
      typ   bin   ;run under BASIC.SYSTEM
      use   gendraw.macs
      org   $4000  ;start above graphics
*---------------------------------------
Graphics    = $c050     ;graphics mode on
MixOff      = $c052     ;no text at bottom
HiRes       = $c057
Wait        = $fca8     ;nifty ROM routine

*
ShapeAddr   = 10  ;direct page space, dude
ScreenAddr  = 12  ;more direct page space
*
ShapeHeight = 6   ;ht (in lines) of shape
ShapeWidth  = 3   ;width (in bytes) of
                  ;shape
*---------------------------------------

ProgramStart
      lda   Graphics    ;turn on graphics
      lda   MixOff      ;no split screen
      lda   HiRes       ;we do want HiRes
      jsr   ClearScreen ;make all black
*
* now draw the word 'HELLO' on the
* screen...twice
*
      DrawShape #HShape;#16;#1;#10;#10
      DrawShape #EShape;#16;#1;#12;#10
      DrawShape #EShape;#16;#1;#12;#30
      DrawShape #LShape;#16;#1;#14;#10
      DrawShape #LShape;#16;#1;#14;#50
      DrawShape #LShape;#16;#1;#16;#10
      DrawShape #LShape;#16;#1;#16;#70
      DrawShape #OShape;#16;#1;#18;#10
      DrawShape #OShape;#16;#1;#18;#90
*
* and now make the little box shape travel
* across the screen
*
      lda   #120
      sta   :Y  ;init vertical coordinate
      lda   #0
      sta   :X  ;and the horiz coordinate
      stz   :ShapeOffset  ;which of 7
                  ;shapes to start with

]loop
      ldy   :ShapeOffset
      lda   ShapeTable+1,y  ;get high byte
      pha
      lda ShapeTable,y      ;get low byte
      pha
      PushByte #ShapeHeight;shape ht
      PushByte #ShapeWidth ;width in bytes
      ldx   :X    ;X coordinate
      ldy   :Y    ;Y coordinate
      jsr GenDraw       ;go draw the shape
*
* now point the offset to the next shape.
* if all 7 shapes have been drawn, start
* over with shape number zero.
*
      lda   :ShapeOffset
      clc
      adc   #2    ;move to the next shape
      cmp   #14   ;have we done them all?
      blt   :NotAll
      inc   :X    ;move to next screenbyte
      lda   #0    ;start with 1st shape

:NotAll
      sta   :ShapeOffset
      lda   #100
      jsr   Wait  ;pause to slow action
      lda   :X    ;see how far we've gone

* are we at the end of the screen?

      cmp   #39-ShapeWidth    ;
      bne   ]loop  ;if not, keep traveling

* quit back to BASIC w/graphics screen
* still showing.
      rts
:X    ds    1
:Y    ds    1
```

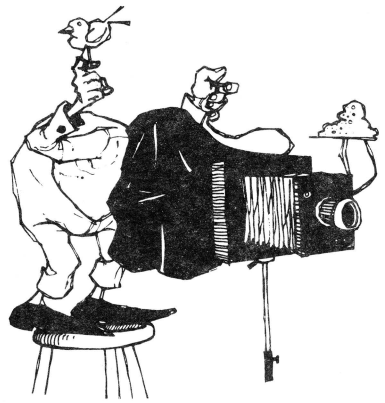
```
:ShapeOffset ds 1

*=======================================
* plot black stuff on the hires screen to
* clear it. Yeah, there's a ROM call for
* this, but we can use this later.

ClearScreen
      lda   #0
      sta   ScreenAddr
      lda   #$20
      sta   ScreenAddr+1 ;start at $2000
]CLoop1
      ldy   #0
      lda   #0 ;plot a zero - it's black!
]CLoop2
      sta (ScreenAddr),y ;plot a black dot
      iny
      bne   ]CLoop2
      inc   ScreenAddr+1
      lda   ScreenAddr+1
      cmp   #64
      blt   ]CLoop1
      rts

*=======================================
* the Generic Drawing routine
*  enter:
*        PushWord ShapeAddress
*        PushByte ShapeHeight
*        PushByte ShapeWidth
*        ldx      XCoord  ;byte pos 0-39
*        ldy      YCoord  ;line pos 0-189
*        jsr GenDraw

GenDraw
      sty   :YCoord     ;save vert coord
      stx   :XCoord     ;save horiz coord
      plx
      ply  ;hold return address for a sec
      PullByte :ShapeWidth
      PullByte :ShapeHeight
      PullWord ShapeAddr;where shape lives
      phy
      phx         ;restore return address

]loop
      ldy   :YCoord   ;vertical position

* look up lo byte of line

      lda   LoTable,y
      clc
      adc   :XCoord ;add offset into line
      sta   ScreenAddr
      lda   HiTable,y
      adc   #0    ;add carry if necessary
      sta   ScreenAddr+1;save it, too.

      ldy   #0

;how many bytes to draw on this row
      ldx   :ShapeWidth

]DrawLoop
      lda   (ShapeAddr);byte of shape tbl
      sta   (ScreenAddr),y;stick on screen

;point to next byte of shape table
      inc   ShapeAddr

      bne   :SkipHi;<>0, don't inc hi byte
      inc   ShapeAddr+1;else inc hi byte

:SkipHi
      iny         ;next position on screen
      dex         ;one byte down...
      bne   ]DrawLoop ;if not done, cont

;if done with all bytes, next line
      inc   :YCoord
;another line bytes the dust
      dec   :ShapeHeight
;if more lines to draw, go do it
      bne   ]loop
      rts         ;udderwise. skedaddle

:ShapeHeight ds   1
:ShapeWidth ds    1
:XCoord     ds    1
:YCoord     ds    1

*=======================================
* address for the preshifted shapes, and
then the shapes themselves.

ShapeTable
      da    Shape1
      da    Shape2
      da    Shape3
      da    Shape4
      da    Shape5
      da    Shape6
      da    Shape7

Shape1 hex    0000007E3F002220002220007E3F00000000
Shape2 hex    0000007C7F004440004440007C7F00000000
Shape3 hex    000000787F0108010108010178 7F01000000
```

```
Shape4 hex     000000707F0310020210020270 7F03000000
Shape5 hex     000000607F0720040420040460 7F07000000
Shape6 hex     000000407F0F400808400808407F0F000000
Shape7 hex     000000007F1F001110001110007F1F000000

HShape hex     636363636363637F7F63636363636363
EShape hex     7F7F03030303031F1F03030303037F7F
LShape hex     030303030303030303030303037F7F
OShape hex     3E7F6363636363636363636363637F3E

*=============================================
* lookup tables so we can find the address of each
screen line.

HiTable hex    2024282C3034383C2024282C3034383C
       hex     2125292D3135393D2125292D3135393D
       hex     22262A2E32363A3E22262A2E32363A3E
       hex     23272B2F33373B3F23272B2F33373B3F
       hex     2024282C3034383C2024282C3034383C
       hex     2125292D3135393D2125292D3135393D
       hex     22262A2E32363A3E22262A2E32363A3E
       hex     23272B2F33373B3F23272B2F33373B3F
       hex     2024282C3034383C2024282C3034383C
       hex     2125292D3135393D2125292D3135393D
       hex     22262A2E32363A3E22262A2E32363A3E
       hex     23272B2F33373B3F23272B2F33373B3F

LoTable hex    0000000000000000808080808080808 0
       hex     0000000000000000808080808080808 0
       hex     0000000000000000808080808080808 0
       hex     0000000000000000808080808080808 0
       hex     2828282828282828A8A8A8A8A8A8A8A8
       hex     2828282828282828A8A8A8A8A8A8A8A8
       hex     2828282828282828A8A8A8A8A8A8A8A8
       hex     2828282828282828A8A8A8A8A8A8A8A8
       hex     5050505050505050D0D0D0D0D0D0D0D0
       hex     5050505050505050D0D0D0D0D0D0D0D0
       hex     5050505050505050D0D0D0D0D0D0D0D0
       hex     5050505050505050D0D0D0D0D0D0D0D0

*=============================================

       sav   gendraw
```



IIgs Programming

Pascal Pics Part II



by Phil Doto

In a previous *8/16* article [April 90], I discussed the care and feeding of Super High Resolution pictures that have been stored in Apple's favorite format ($C0/$0002). Apple Preferred Format files are very flexible and can be used to store any SHR picture; unfortunately, not all IIGS pictures are in this format. If you want your program to display pictures that someone else created, you'll have to work with what you get. Also, even though I happen to like APF files, I have to admit that in some cases (for example, a screen sized image) there are advantages to using another format. So, in the spirit of fair play and completeness, it's time to talk about the other common SHR file types.

Super Hi-Res pictures come in two main file types. Type $C0 (commonly called PNT files) are compressed picture files and type $C1 (PIC files) are uncompressed pictures. The auxiliary file type is used to specify the exact type of file within each category. For example, a compressed file ($C0) with an auxtype of $0002 is an Apple Preferred Format file.

For pictures that do more than just sit there and look pretty, there's a third file type. Type $C2 (ANI files) are animation images.

Paintworks Pictures

The first auxiliary type ($0000) for compressed ($C0) files was assigned to files created by one of the first products to work with SHR files, Activision's Paintworks program.

Before we start digging around inside a Paintworks file, let's quickly review the data we need to display an SHR picture. In my first article, I discussed the following data structure:

```
type PicRecord =
       record
          ImageHandle :        handle;
          MasterMode :         integer;
          PixelsPerScanLine :  integer;
          NumScanLines :       integer;
          LineSCB : array[0..MaxLine] of integer;
          NumPalettes :        integer;
          Palette : array[0..15] of ColorTable;
       end;
```

This record contains all the data needed to properly display an SHR picture. If I load the file, unpack the picture, and fill out this record, I can use the same routines that I used with the APF files to display the picture on the screen. This is an easy way to support several different file types in the same program. Once the data is in this record, it doesn't matter what the original file type was.

All we need to know now is how the data is stored in the Paintworks file and we can get started. According to Apple's file type note on $C0/$0000, Paintworks files contain the following:

Byte	Contents
+000 to +01F	Super Hi-Res Palette
+020 to +021	Background Color
+022 to +221	16 QuickDraw II patterns
+222 to EOF	Packed graphics data

The background color and patterns are needed by the paint program, but they aren't needed to display the picture, so let's ignore them. That leaves a palette and a packed image -- not enough data to fill out the PicRecord. Of course, the answer is that the other fields in the record are the

same for all Paintworks files. These files are 320 mode pictures, so the MasterMode and all the Scan Line Control Bytes will be $00. They are screen wide images, so there are 320 pixels per scan line. Finally, the pictures are "page" length with 396 scan lines.

Although $C0/$0000 files are always 320 mode pictures, 640 mode Paintworks files do exist. The 640 mode pictures use an auxiliary file type of $8000. The file contents are the same as the $0000 files, but the $8000 auxtype tells us to use 640 mode for the display. When I first started experimenting with these files, I was a little surprised to learn that, although the MasterMode was $80, the scan line control bytes still needed to be $00.

Listing 1 is a routine to load, interpret, and unpack both types of Paintworks files. This is a parallel routine to the LoadAPF routine in the prior article. When I load a $C0 file, I can check the auxtype and call LoadAPF for auxtype $0002 or call this routine for auxtypes $0000 or $8000.

```
  {*---------------LISTING 1-----------------*}

function LoadPW(thePathName : GSString255;
                VAR thePic:PicRecord;
                theAuxType:longint): boolean;

    { This function loads a PaintWorks     }
    { file specified by the GSOS string.   }
    { If successful, true is returned and  }
    { all fields of the PicRecord are set. }

var  myBufferHndl : Handle;
     i,
     bytesDone,
     srcSize,
     dstSize : integer;
     srcBuffer,
     dstBuffer : Ptr;
     address,
     size : longint;
begin
  LoadPW := false;

  { load the file and get the address }

  if LoadFile(thePathName,
              myBufferHndl,
              size) then begin

    address := longint(myBufferHndl^);

    with thePic do begin

      { 396 scan lines with SCB = $00 }

      NumScanLines := 396;
      for i := 0 to 395 do LineSCB[i] := $00;

      { one palette at start of file }

      NumPalettes := 1;
      Palette[0] := ColorTablePtr(address)^;

      { get mode from auxtype }

      if theAuxType = $0000 then begin
         MasterMode := $00;
         PixelsPerScanLine := 320;
         end
      else begin  {must be $8000}
         MasterMode := $80;
         PixelsPerScanLine := 640;
         end;

      { allocate space for $F780 bytes }
      {   160 X 396 = 63360 = $F780    }

      ImageHandle := NewHandle($F780,
                               myMemoryID,
                               attrLocked,
                               nil);
      theError := _ToolErr;
      if theError <> noError then begin
         ReportError(theError);
         DisposeHandle(myBufferHndl);
         exit(LoadPW);
         end;

      { compressed image is at offset +$222 }

      srcBuffer := Ptr(address + $222);
      srcSize := size - $222;

      dstBuffer := ImageHandle^;
      dstSize := $F780;

      bytesDone := UnPackBytes(srcBuffer,
                               srcSize,
                               dstBuffer,
                               dstSize);

      DisposeHandle(myBufferHndl);

    end; {with}

  LoadPW := true;
  end;
end;

  {*-----------------------------------------*}
```

This routine first loads the file image into memory using the same procedure that I used for APF files. Next, appropriate values are assigned to the fields of the PicRecord. The palette is extracted from the file and the other fields are assigned the appropriate fixed values depending on the auxtype.

Unpacking the image is simpler than it was with APF files since I don't have to deal with all kinds of odd picture sizes. The unpacked image size will always be 160 bytes per scan line times 396 scan lines which is 63360 ($F780) bytes. All I need to do is allocate the memory, set up a few parameters and feed them to UnPackBytes.

Once the picture is unpacked, I can dispose of the memory that was used for the file image and return true to the calling procedure to indicate that the PicRecord contains a pixel image and other data ready for the display routines.

Screen Images

Conceptually, the simplest type of SHR file is an image of the 32K screen display in computer memory. Screen images come in two flavors, compressed ($C0/$0001) and uncompressed ($C1/$0000). A $C0/$0001 file is simply a $C1/$0000 file that has been compressed by processing the entire file with PackBytes. Likewise, if we run a $C0/$0001 file through UnPackBytes, we end up with a $C1/$0000 file.

Although conceptually very simple, screen images often cause confusion. The source of most misunderstandings about these files is the mistaken idea that they simply consist of the pixel image. In fact, a 32K screen image includes the pixel image, the palettes, and the scan line control bytes.

Figure 1 - Screen Image File Contents

Memory Locations	File Bytes	Contents
$E1/2000 - $E1/9CFF	$0000 - $7CFF	pixel image
$E1/9D00 - $E1/9DC7	$7D00 - $7DC7	200 SCBs
$E1/9DC8 - $E1/9DFF	$7DC8 - $7DFF	resrvd (zeros)
$E1/9E00 - $E1/9FFF	$7E00 - $7FFF	16 palettes

A simple way to display this type of file is to load it directly back to $E1/2000 (after processing it with UnpackBytes if it's a compressed file.) That way the scan line control bytes and the palettes will automatically fall into place and the picture will be displayed properly.

However, sometimes one wants to store the image off screen, so the image (or parts of it) can be copied to the screen (or a window) at a later time. Off screen processing is also handy in the case of a program that is supporting several different file types. Cases like these call for an approach similar to that used for the other file types.

Listing 2 is a routine to load, interpret, and unpack both compressed and uncompressed screen image files. Once again, this is a parallel routine to the other routines I've discussed.

```
{*---------------LISTING 2-----------------*}

function LoadImage(thePathName : GSString255;
                   VAR thePic:PicRecord;
                   theType:integer): boolean;

    { This function loads a screen image    }
    { specified by the GSOS string.         }
    { If successful, true is returned and   }
    { all fields of the PicRecord are set.  }

var  myBufferHndl : Handle;
     i,
     bytesDone,
     srcSize,
     dstSize : integer;
     srcBuffer,
         dstBuffer : Ptr;
         address,
         size : longint;
   begin
     LoadImage := false;

     { load the file }

     if LoadFile(thePathName,
                    myBufferHndl,
                    size) then begin

     { and fill in the pic record }

        with thePic do begin

      if theType = $C0 then begin

         { packed file - first allocate 32K }

         ImageHandle := NewHandle($8000,
                                  myMemoryID,
```

```
                              attrLocked,
                              nil);
   theError := _ToolErr;
   if theError <> noError then begin
      ReportError(theError);
      DisposeHandle(myBufferHndl);
      exit(LoadImage);
      end;

   { then unpack the whole file }

   srcBuffer := myBufferHndl^;
   srcSize := size;

   dstBuffer := ImageHandle^;
   dstSize := $8000;

   bytesDone := UnPackBytes(srcBuffer,
                              srcSize,
                              dstBuffer,
                              dstSize);

   DisposeHandle(myBufferHndl);
   end

else   { unpacked file }
   ImageHandle := myBufferHndl;


{ get the address for pointer math }

address := longint(ImageHandle^);


{ 200 SCBs starting at offset +$7D00 }

NumScanLines := 200;

for i := 0 to 199 do
   LineSCB[i] := intPtr(address
                        + $7D00 + i)^;


{ 16 palettes starting at offset +$7E00 }

NumPalettes := 16;

for i := 0 to 15 do Palette[i] :=
   ColorTablePtr(address
                 + $7E00 + (32 * i))^;


{ get mode from first scan line }

if BAnd(LineSCB[0],$80) = 0

   then begin   { 320 mode }
           MasterMode := $00;
           PixelsPerScanLine := 320;
           end

        else begin   { 640 mode }
                MasterMode := $80;
                PixelsPerScanLine := 640;
                end;

     end; {with}
  LoadImage := true;
  end;
end;

   {*-------------------------------------*}
```

The LoadImage routine first loads the file into memory using the same LoadFile function. Next, if it's a compressed file (type $C0), 32 kilobytes are allocated and the file is unpacked. Since an unpacked $C0/$0001 file is exactly the same as a $C1/$0000 file, the rest of the processing is the same.

Since the pixel image is the first thing in the file, I can use the handle to the file image as my ImageHandle. The pixel image is only $7D00 bytes of the $8000 handle, so I could resize the handle after extracting the other data, but it isn't necessary. Also, leaving the extra information behind the image has an advantage that I'll explain shortly.

It's a fairly simple matter to fill in the fields of the PicRecord. The SCBs and palettes come from the file and most of the other information is fixed since this is a screen sized picture. Since the MasterMode isn't explicitly stated, I've taken it to be the same as the mode of the first scan line control byte.

Animation Files

Paintworks animation files (type $C2) provide a way to store pictures that move. Although listed as an official file type assignment, there is no file type note on $C2. In fact, as far as I know, the file structure has never been published anywhere. The following file structure is inferred by inspection (with the help of *8/16*'s bright, young IIGS editor, one Eric Mueller):

Paintworks Animation File Format:

Byte	Contents
+0000 to +7FFF	Screen image of first frame
+8000 to +8003	Length of animation data block
+8004 to +8007	Delay time
+8008 to EOF	animation data block

The first thing in the file is an ordinary uncompressed screen image of the first frame to be displayed. This is exactly like a $C1/$0000 file. As a result, I can get double duty out of the above LoadImage routine and use it to load these files as well. Since I left the whole file image in the ImageHandle, the animation data will be in the PicRecord behind the pixel image.

After the screen image, we find two 4-byte values. The first is the length of the animation data block and the second is a delay value that tells how long to pause between frames. Although four bytes are allocated for this delay value, the numbers are small and only one is apt to contain non zero data. A delay of one tick (1/60 second) times this value seems to work out about right. In other words, a delay time of 6 will result in a tenth of a second delay between frames.

Finally, the animation data block tells how to modify each frame to create the next one. This block starts out with a 4-byte value. Since on the files that I've examined the value is always 4, I think that this was an offset to the actual data. It was probably included to allow for future expansion. The actual data consists of a series of four byte records. Each of these records is made up of two 2-byte values. The first is an offset into the display screen. Adding $E12000 to this offset will give an address in screen memory. The next two bytes contain pixel data to poke onto the screen at this address. Each frame consists of the pixels needed to modify the prior frame. An offset of zero indicates the end of a frame.

In order to display the animation, I load the file with LoadImage and display the first frame just like it was a $C1/$0000 file. Then I call this routine to animate the image.

```
   {*---------------LISTING 3-----------------*}

procedure DoAnimation;

     { This function animates a $C2 file }
     { in the global PicRecord myPic.    }

type longintPtr = ^longint;

var  theEvent :  EventRecord;
     StopAni : boolean;
     code,delay,
     offset,pixelData : integer;
     DataPtr,AniBlockLength,EndTick,
     StartAddress,EndAddress,
     WriteAddress : longint;

begin
   StopAni := false;  { initialize a flag }

   { read the length of the ANI block }

   DataPtr := longint(myPic.ImageHandle^)
              + $8000;
   AniBlockLength := longintPtr(DataPtr)^;

   { and the delay value }

   DataPtr := DataPtr + 4;
   delay := intPtr(DataPtr)^;

   { ANI data starts 4 bytes into ANI block }

   DataPtr := DataPtr + 8;
   StartAddress := DataPtr;
   EndAddress := StartAddress
                 + AniBlockLength - 4;

   { keep looping through the animation until }
   { the user hits a key or the mouse button }

   Repeat

     { read the offset number }

     offset := intPtr(DataPtr)^;

     if offset = 0 then begin

        { delay }

        EndTick := GetTick + delay;
        repeat until GetTick > EndTick;

        { and then start a new frame }

        DataPtr := DataPtr + 4;
        if DataPtr >= EndAddress
           then DataPtr := StartAddress;
        end

     else begin

        { we got a non zero offset so }
        { write the pixel data to the screen }

        DataPtr := DataPtr + 2;
        pixelData := intPtr(DataPtr)^;
        DataPtr := DataPtr + 2;
        WriteAddress := $E12000 + offset;
        intPtr(WriteAddress)^ := pixelData;

     end;

     if GetNextEvent($002E,theEvent) then begin
```

```
        { $002E = key or mouse button event }

        if (theEvent.what = keyDownEvt) or
           (theEvent.what = autoKeyEvt)
           then begin

           code := LoWord(theEvent.Message);
           case code of

            { increase delay on down arrow }

              10 : delay := delay + 1;

            { decrease delay on up arrow }

              11 : if delay > 0
                     then delay := delay - 1;

            { stop animation on any other key }

              otherwise StopAni := true;
           end; {case}
        end  {if key event}

        { stop animation on mouse button }

        else StopAni := true;
      end;

   until StopAni;

   ClosePic(PicLoc);
end;

   {*-----------------------------------------*}
```

The routine determines the starting and ending addresses for the animation data and reads the delay time before entering the animation loop. The animation loop is really quite simple. Read an offset and, if it is not zero, calculate the screen address, read the pixel data and poke it onto the screen. If the offset is zero, pause for delay ticks and then update the data pointer and start the next frame.

Each time through the animation loop, I check to see if a key or mouse button event has occurred. If a down arrow was pressed, increment the delay to slow the animation down. If an up arrow was hit, decrement the delay and the animation speeds up. If the user presses any other key or the mouse button, I stop the animation and return to the desktop display.

Putting it all together

Combine these routines with the routines from the previous article and you'll be able to display any common SHR graphic file. Since, in all cases, the file image is saved offscreen; all sorts of special effects are possible. You could stretch the image with CopyPixels or make a puzzle by copying parts of the image. The possibilities are endless, but one thing is certain. Whether he or she is going to write the next great paint program or just wants to add a title screen, sooner or later every IIGS programmer will use SHR picture files.

From the House of Ariel

The Great 8/16 Early Renewal Special !!!

❒ First, the deal you've *all* been asking for (okay, so we were a little slow on the uptake...). If you renew and order *both* the magazine and the disk, you get to take $10 right off the top of your total.

❒ Second, if your renewal is postmarked before midnight December 31st, 1990, you get to renew at the old $29.95 price (for the magazine). This is $2.05 off the normal subscription price we will have effective January 1st, 1991 (in fact, our official price is already $32, but we've had some intro offers on direct mail ads through the end of this year). Okay, I admit that $2.05 isn't a world class savings, but if you combine it with the $10 off the disk/magazine combo, you'll get nearly 15% of the normal price.

❒ Third, if you renew before the end of the year, you may choose to order SSSi's DeskPak™ GS desk accessory package for a mere $15 (about 60% off!). This super package contains 18[1] different DA's for your GS, including an appointments reminder, file tools (no more having to quit back to the finder to delete a file!), a calculator, mini-database, screen saver, and two different scrap books. For those of you who aren't familiar with a scrapbook DA, they are the biggest productivity tool for the desktop environment. They let you store text or graphics in such a manner that that they're always available from the DA. Select what you want and you can paste it into any document - anytime, anywhere. Best of all, you'll receive a huge price break when SSSi updates the package! Oh, I almost forgot: **if you order DeskPak from us, we'll make your current subscriptions - both disk and magazine if applicable - go an extra month.** This is kinda like getting a rebate at the time of purchase. This offer applies to both renewals and DeskPak-only orders.

We have included a special order form on the back cover of this month's issue.

• 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

1. Please note that one of the DA's ("Add DA") has become incompatible with current system software. This will be rectified in the new updated version of the package, due out in early '91. All of the other DAs perform as advertised on my GS (a ROM 01 machine running System 5.02). - Ross

"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.

A few highlights (so far every disk has had more than 600K of material!):

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

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

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

- **June '90:** 8 bit - 3D graphics package, MicroDot™ Demo, DiskWorks, 80 column screen editor.

  16 bit - Assembly Source Code Converter (shareware), Install DA (on the fly; by our our own Eric Mueller), Find File source code.

1 year - $69.95 **6 months - $39.95** **3 months - $21**

Individual disks are $8.00 each. Non-North American orders add $15 for 1 year, 8$ for 6 months, and $5 for three months. All disks are shipped first class.

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

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

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

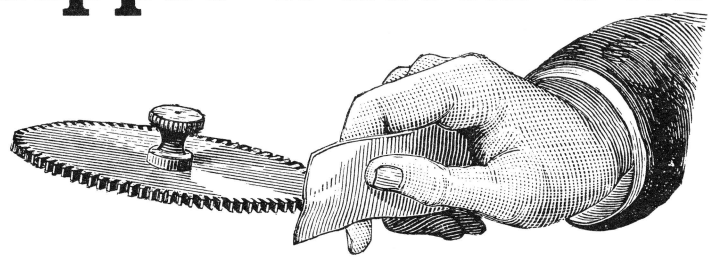
Our guarantee: Ariel Publishing guarantees your satisfaction with our entire product line (software and publications). If you are ever dissatisfied with one of our products, we will cheerfully refund the amount you paid on your request. To order, just write to: **Ariel Publishing, Box 398, Pateros, WA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

The Weekend Hardware Hacker

Giving Your Apple a Real Switch

by David Gauger
Oral Roberts University



(Frankly, I've been amazed at the interest in David's columns. Not only have folks written in to say how much they've enjoyed using the projects after building them, but Don Lancaster even gave us a plug in his Radio-Electronics magazine column. Many of our new subscribers are therefore dedicated project builders. If that was not enough, one of David's future productions for us is already being incorporated into a commercial product.

This month's foray is a littler simpler than past projects - partly at my request. Things couldn't have worked out better though - David gave us a switch project in the very month Roger Wagner released HyperStudio XCMDs that respond to switch-type input devices(c.f. my Publisher's Pen column). Perhaps one of y'all will design some hardware and software for your local special education department, etc. If you do, I'd like to hear about it. -- Ross)

With a little imagination and just a bit of switch hardware, your Apple can sense and react to the world around it in some amazing ways. Because the gameport makes connecting switches and sensors a trivial task, the basic Apple][is equipped far better than many other computers to interact with the outside world. Even so, few features of the Apple][line have as much unrealized potential as the game port switch inputs.

In this article we'll discuss different types of switches, how to connect them to the gameport, and what they can be used for. We'll also take a look at some devices that you probably never thought were switches. The objective is to introduce concepts and capabilities, not a finished program or system. Hopefully these ideas will stir your own creativity and prompt you to see applications for the various devices we'll look at here.

All the parts used are inexpensive, available from Radio Shack, and involve a minimum of soldering. The basic switch circuit in this article will work on any Apple from the earliest Apple][to the most advanced GS. If you've never built a hardware project before, this is a great place to start *(Thank you. I will. - Ed)*.

A Switch is a Switch is a Switch

There really isn't much to a switch. It's sole function is to make or break a connection between two or more wires. Obviously, these wires carry electricity. Many people are afraid to experiment when electricity is involved, but the voltages we will be working with in this article are only 5 volts at the most. This is not enough voltage to even feel, let alone be dangerous.

Electricity must flow in a circle. This is why collections of electronic components are called circuits: in order to work, the electricity must flow completely around the circuit in a\more or less circular fashion. If the circuit path is broken at any point, the electricity stops flowing and the circuit stops functioning. One of the basic uses of a switch is to control when a circuit functions (circuit completed) and when it doesn't (circuit broken). The on/off switch on your Apple][is an example.

There are two kinds of electricity: alternating current (AC) and direct current (DC). The electricity used in the game port is DC which means it always flows in one direction. This gives it a polarized nature involving positive and negative charges. For our discussion, what flows where is

not as important as the idea that electricity flows in one direction from one point to another in a complete and unbroken circuit.

Poles and Throws

There are many kinds of switches and nearly as many ways to categorize them. Two basic categories are the number of "poles" and the number of "throws." A pole corresponds to an electrical connection. If a switch has a single pole, a single connection is made or broken when you operate it. Many wall light switches are single pole devices: flipping the switch causes one light or circuit to be turned on or off.

In a double pole switch two connections are made or broken simultaneously when the switch is activated. This makes it possible to complete two separate circuits at the same time with one flip of a switch. A double pole switch acts like a pair of single pole switches operated simultaneously by one handle.

Throwing the Switch

Another category is the number of throws a switch has. A single throw switch either connects or disconnects one wire to another. In other words, flipping the switch in one direction makes the connection and activates the circuit. Flipping it in the other way breaks the circuit thereby turning it off.

A double throw switch behaves differently. It works by taking an incoming wire and connecting it to one of two outgoing wires. It's essentially a selection device with two possible outcomes.

Flipping the switch one way connects the input to wire "A" while flipping it the other way connects the input to wire "B". You can route the incoming signal to either wire "A" or wire "B" but not both at the same time. Obviously, if you only utilize one of the two throws in your circuit, you can make a double throw switch act like a single throw switch.

A double throw switch will have at least three terminals for wire connections. Usually, the center terminal will be the wire that will be connected to one of the other two terminals, depending on how the switch is currently activated. A double pole/double throw switch will have six terminals.

If you're in doubt as to which terminals get connected to what internally, the best bet is to get a multi-meter or a meter that has a resistance or continuity checker built in. You should measure very little resistance when the contacts are closed (less than 1 ohm) and an (almost) infinite resistance when they are open. Obviously, when a switch is closed the wires it controls are connected. When it is open the wires are disconnected. By checking the resistance between the terminals when the switch is in various positions you can determine which terminal is connected to which internal contact. See Fig. 1 for the internal schematic of several switches.

A switch can be defined by its poles and throws classification. Switches can have any number of poles and throws. Rotary switches often have quite a few of each. Toggle switches usually have just a few, although I have seen a 4 pole, double throw toggle switch. Common switches have only a couple of each. Most off-the-shelf switches fall in one of four categories: single pole/single throw, single pole/double throw, double pole/single throw (not common), and double pole/double throw.

While almost all of these types are available from your local Radio Shack store, a look in the catalog reveals that the classifications are abbreviated to one letter each. A double pole/double throw switch is abbreviated to a DPDT switch while a single pole/single throw is listed as a SPST switch. The switches most often used with the gameport are of the SPST variety.

Just a Moment, Please

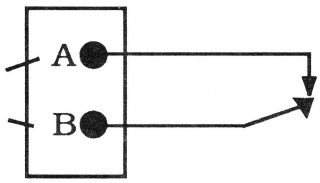
Some switches contain a spring which returns the switch to its original state after actuation. These switches are called "momentary" because they are activated only while the user presses or manipulates it. A typical example is the fire button on a joystick. Obviously, the fire button is activated (on) only while the button is pushed by the user. After that, the spring returns the switch to its original resting state (off).

A momentary switch is constructed so that its normal state is either "open" or "closed". A switch is closed if it is currently connecting the two wires, but "open" if the wires are disconnected. Pressing the switch reverses its state. For example, a normally open (abbreviated N.O.) switch will be open if you do nothing to it. If you activate it, you'll close the switch. The opposite is true with a normally closed (N.C.) switch: it's normal state is closed, but it will open if you press it. The fire button on your joystick is a momentary N.O. SPST pushbutton switch. Try that line out on your friends!

Figure 1 - Common Switches

Single Pole/Single Throw

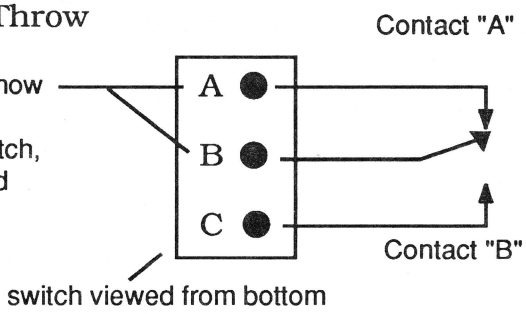
Contacts are not polarized so either terminal can go to 5 volts or ground



Contacts are currently touching so terminals "A" and "B" are connected. Flipping the switch the other way will disconnect the terminals.

Single Pole/Double Throw

Terminals "A" and "C" are now connected internally by the contacts. If you flip this switch, terminals "B" and "C" would connect.

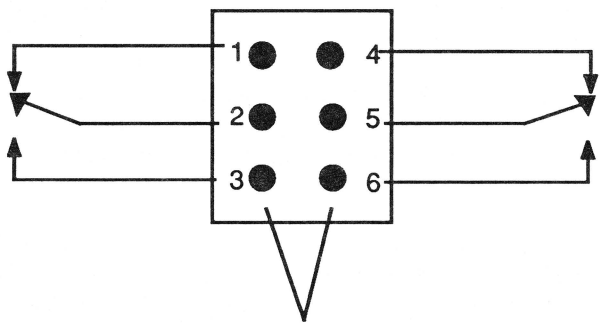


Internal movable contact "C" switches between stationary contacts "A" and "B"

Double Pole/Double Throw

This switch is currently conecting terminals 1 & 2 and separately connecting terminals 4 & 5.

If flipped the other way, this switch would connect terminals 2 & 3, and in a separate circuit, 5 & 6.



Note that there is never any internal connection between terminals 1 & 4, 2 & 5, or 3 & 6. Terminals 1, 2 & 3 act like one switch, and terminals 4,5, & 6 act like another.

About Those Amps and Volts

Within a switch, the actual opening and closing of the circuit is accomplished by a set of contacts. The size and construction of these contacts determines how much voltage (electrical pressure)

and amperage (electrical current) a switch can handle. Obviously, if you exceed the contact's ratings in either category, there can be serious consequences. There is no danger with the gameport, though. The voltage and amperage are so low you can connect to it any switch that Radio Shack sells with no problem at all.

Connecting the Switches to Your Apple

This next section is about as technical as we'll get in this article. For those who are not technically minded, or haven't built many electronic projects, stay with us through this section and in the end you'll be able to connect just about any switch you can think of to your Apple.

The procedure is very simple because of the gameport. The pushbutton inputs need only a single resistor in addition to the switch to complete the circuit. Fig. 2 shows the schematic diagram of how to connect it up using switch 0. Notice that there are two diagrams. This is because there two different connectors used for the joystick on various Apples. For Apple][,][+, use the schematic for the 16 pin DIP. For the //c, //c+ use the DB-9 schematic. //e and GS owners can use either one. The schematics are identical except for the connector.

For those who don't read schematics, the resistor goes from the gameport ground pin to the pin of the switch input you want to use, in this case, switch 0. Often, I just clip the resistor's leads quite short and mount the resistor right on the gameport connector, whether it's a DB-9 or 16 pin DIP. If you're using a SPST switch, just connect the two switch terminals to the 5 volt terminal and the switch input respectively. Switches are not polarized in any way, so it doesn't matter which switch terminal goes to the ground pin and which goes to the pin supplying 5 volts.

Pull-Down Resistors

(Non-"Techies" Can Skip the Next 2 Paragraphs!)

The resistor needed to interface the switch to the gameport is called a "pull-down" resistor. Inside your Apple, the pushbutton input lines are connected to a chip that belongs to the TTL (transistor-transistor-logic) family. Unconnected pins on a TTL chip float on their own accord up to 5 volts. This creates a problem since your Apple thinks any pushbutton input at 5 volts is "pushed". In essence, if you leave the pushbutton input unconnected, your Apple will think the switch is always pushed.

The solution is to use a resistor to pull the voltage on that pin down near ground level (0 volts). Note that the resistor goes from the pushbutton input pin to ground making a connection through the resistor to ground potential. The Apple then sees 0 volts on the pushbutton input which it interprets as an open (unpushed) switch. When you push the switch, the contacts close connecting the pushbutton input directly to the 5 volt supply pin forcing it back up to 5 volts despite the pull-down resistor. Obviously, when this happens the button appears pushed to your Apple. When you release it, the pull-down resistor does its job and pulls the pushbutton input back down near 0 volts.

Hardware information sources (including the Apple technical manuals themselves) recommend various sizes for the pull-down resistor. The resistor in my commercially made joystick measures 560 ohms which is somewhere in the middle of the range that the various manuals recommend. The joystick works well on a][+, //e, and //GS. It should also work fine on the //c and //c+, although I have not had a chance to try it. For your switch projects, start with a 470 ohm (Cat. #271-1317) or 560 ohm (#271-020) pull-down resistor. If it doesn't work for any reason, try either a 680 or 330 ohm resistor. The value of this resistor is not extremely critical.

A look at the pinout diagrams for the various Apple][']s reveals that different models have been endowed with varying numbers of pushbutton inputs. The //c and //c+ have 2, the][,][+, and //e have 3, and the GS has 4. They are all wired the same way: a wire from the switch goes to the +5 volts supply, and the other goes to the pin of the switch input you desire. A resistor goes from the switch input pin to ground. This may mean that you have 3 or 4 wires all connected to the 5 volt pin at once, and just as many resistors connected to the ground pin, but it doesn't matter: your Apple will handle this just fine.

The Software Side of Things

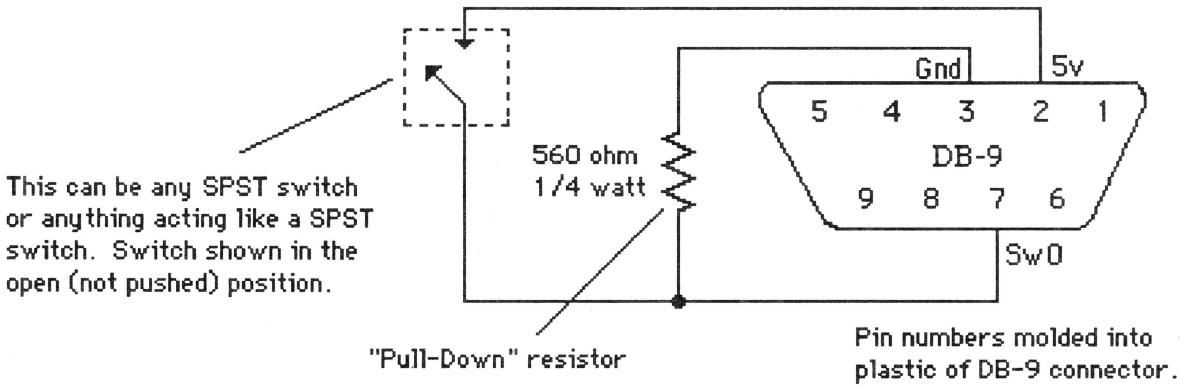
The Apple is set up so that each pushbutton input has a pin in the gameport and its own address in memory. For example, address 49249 (hex $C061) is pushbutton #0. (Computers number things starting with 0.) Address 49250 is pushbutton #1, and, if you have an Apple other than a //c or //c+, 49251 is pushbutton #2. On the GS, address 49248 is pushbutton #3.

To read a pushbutton, you just read it's address. In Applesoft, this can be accomplished with a PEEK

Figure 2 - Paddle Input Switch Connections

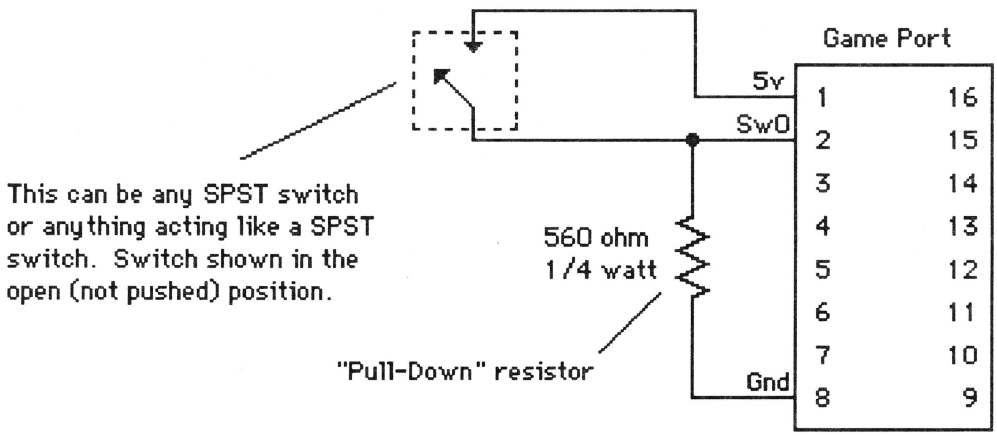
Schematic Diagram - DB-9

Use with Apple *IIc, IIc+, IIe, IIgs*



Schematic Diagram - 16 Pin DIP

Use with Apple][,][+, *IIe, IIgs*



Note: The 16 Pin DIP connector is no longer available from Radio Shack. It is available from many electronics parts sources such as Digikey and Jameco. Jameco's part number is 16HP. Jameco Electronics, 1355 Shoreway Road, Belmont, CA 94002 (415) 592-8097

instruction. For example, to read pushbutton 0, you could use the following code:

```
X=PEEK (49249)
```

This instruction sets the variable X equal to the pushbutton value. The value of X can be anything from 0 to 255. This is because all Apple][‘s use 8 bit memory and the largest value 8 bits can hold is 255.

The switch is actually connected to only one of the 8 bits in the byte you read: the most significant bit. This bit has a value of 128 in the binary numbering system, whereas the sum of all the other bits in the byte amounts to only 127. Detecting a pushed button is a simple matter of math: if the value of the byte is 128 or more, the button is pushed. If the value is 127 or less, the button is not pushed.

From Applesoft, you can detect a button push easily. For example, the following program is an endless loop that beeps the speaker whenever button #0 is pushed. Otherwise it just goes around the loop doing nothing.

```
10 IF PEEK(49249) > 127 THEN PRINT CHR$(7);
20 GOTO 10
```

Detecting a button push from assembly language is equally easy. Perhaps the simplest way to do it is to use the BIT instruction which passes the bit activated by the switch into the negative flag of the status register. If you push the button while the BIT instruction is executed, the negative flag in the status register will be set to one. There you can check it with the instructions BMI (branch on minus) or BPL (branch on plus) instructions. A BMI instruction will branch when the button is pushed.

The following assembler language fragment does exactly what the Applesoft code does above:

```
LOOP BIT  $C061   ;$C061 is button #0
     BPL  LOOP    ;button not pushed, loop again
     JSR $FBDD    ;$FBDD is the monitor's bell
     JMP  LOOP    ;do loop again
```

Polling Isn't Only for Herefords

The switch input ports look just like normal memory to the 6502, 65C02, or 65816 microprocessors. In fact, this family of microprocessors treats all I/O (input or output) as normal memory. This is called "memory mapped I/O".

Your Apple's microprocessor (like all others) is not aware of every memory location's activity at the same time. At the most, it can access one memory location at a time. The result is that unless you constantly check the pushbutton's location for activity, your program might miss a button push. Checking a location repeatedly looking for an change of some sort is called polling. Both of the example code fragments above were organized as loops that checked the status of the pushbutton each time around the loop. They are examples of polling the pushbutton inputs.

Polling loops tie up the microprocessor and are not the most efficient way to handle many tasks. It is possible to design hardware to take advantage of interrupts, a feature most microprocessors have. In this situation, when you push the button, a special dedicated input line to the microprocessor is activated signalling that a device (in this case, the pushbutton) needs attention.

Interrupt driven systems do not need to poll the hardware since the microprocessor jumps to a special interrupt handling routine only when interrupted by the special interrupt signal line. This routine typically takes care of what ever the interrupting device needs and then goes back to where it was before the interrupt occurs. This is a much more efficient way to handle a hardware event such as a pushed button.

While an interrupt driven system is certainly possible, it is also much more involved in terms of hardware. Since the point of this column is to keep the hardware simple (so you can build it in a weekend!), we'll use software polling of the switch to know when it is pushed.

Easily Available Switches

The following quick tour through Radio Shack's switch catalog does not constitute a promotion or advertisement. It is just an attempt to show the wide variety of readily available switch types. Where catalog numbers are shown, they are often just one example of that type of switch. Sometimes many different switches of that type are available.

Radio Shack sells a great variety of toggle switches. Many shapes and sizes are represented with SPST, SPDT, and DPDT types common. Some of the more interesting types are momentary (spring loaded return) and three position toggle switches where the center position is off. Using a three position (center off) switch makes it possible to activate

either of two pushbutton inputs (or neither one) with just one switch.

You can also buy a variety of pushbutton switches. Radio Shack has both the push-on/push-off (#275-1555) and momentary types. Momentary switches are available in both the normally open (#275-1547) and normally closed (#275-1548) formats.

Slide and rocker switches are available as are 2 types of rotary switches. There is even an old fashioned DPDT knife switch.

Pushbutton switches need not be activated exclusively by a finger. Suppose you mounted a small (4" by 4") board or plate over 4 pushbutton switches, one in each corner. Pushing anywhere on the plate would then activate one or more switches. Pushing hard enough in the center might even activate all four. This method could yield spacial information about where a person pushed on the plate, not just if he did or didn't.

Quite a few car race games use a joystick's fire button to activate the car's gas feed pedal. Many times I've considered using the N.O. momentary foot switch (Cat. #44-610) for the gas pedal. It would be more stable if mounted on a supporting board of some kind along with a foot switch for the brake pedal. The //e and GS both have 2 different connectors for the game port (16 pin DIP and DB-9). The joystick could use one connector, while these foot switches could be wired to the other.

Other Switches

Catalog #275-027 is a SPDT mercury switch. The ball of mercury conducts electricity. When the angle of the bulb is just right it comes in contact with one of the other wires inside the glass envelope completing the circuit. By positioning the bulb carefully it can be used to detect angles with respect to the ground.

It can also detect motion. This particular mercury switch is a double throw type with connection made when the mercury is at either end. This could be used to sense a reversal of direction due to the inertia of the mercury itself. Quadriplegic individuals often use mercury switches attached to a headband so that a nod of their head activates the pushbutton input.

Catalog number 275-017 is commonly known as a "microswitch". It takes very little pressure (only 5 grams) to operate it. In addition, this model has a 3/4 inch lever. Because of this, it has many specialized applications. For example, some quadriplegics have a microswitch mounted on the frame of a pair of glasses so that the lever can be manipulated by their eyebrow. Many programs have been written using the activation of just one switch to control the entire program. *(And this is now possible for HyperStudio stacks as well. - Ed)*

This particular microswitch has a roller at the end of the lever making it a natural for applications involving a cam of some sort. Because we know the pressure needed to operate it perhaps it could form the basis of a crude scale for extremely light weights. If the switch is closes when an object is placed on the lever, the object obviously weighs more than 5 grams.

Unusual Switches

Radio Shack also carries other items that masquerade as something else, but in reality are switches. For example, the vibration detector (Cat. #49-521) is actually nothing more than a specialized switch. One of its internal contacts is secured to the body of the unit, while the other is mounted on a flexible blade that has a small weight attached to it. The contacts are deliberately adjusted so that they touch very lightly making it a normally closed, single pole/single throw switch.

When a vibration moves the main unit, the inertia of the weight on the moveable blade causes the contacts to make and break connection repeatedly during the vibration. By polling the switch constantly and watching for the first time the contacts break connection your Apple can tell when the sensor detects a vibration. The unit's sensitivity can be adjusted by turning a screw that changes the resting pressure of the contacts. The lighter the pressure, the more sensitive the unit because it takes less of a vibration to cause the contacts to separate.

Home Security Switches

The vibration sensor is part of a whole collection of home security sensors almost all of which are switch type devices. This is because most security systems are designed around a loop of normally closed switches. If all the switches are closed, the circuit is unbroken and all is well. The moment one of the switches opens, (because a door is forced open or a window broken, for example) the burglar alarm senses the open circuit and rings bells, alerts the police, turns on lights, etc.

In the meantime, there is nothing to prevent us from using these sensors as normal switches that

can be connected to the game port just like all the others. As a matter of fact, an Apple would form the basis of an excellent burglar alarm system, but that is a subject for another day!

The glass breakage detector (#49-516) contains a small mercury bulb SPST switch in a housing which enables you to easily adjust the angle of the bulb. You mount the unit on a pane of glass with double faced tape. The idea is to adjust the angle so that the mercury jiggles and makes or breaks contact when the glass is broken. For a normally closed system, the angle is adjusted so that the mercury just barely makes contact all the time. A jiggle breaks the contact for a split second. In a normally open system, the angle is adjusted so that it just barely breaks contact. In this case, a jiggle would cause the mercury to make contact closing the circuit. Either way, this switch can be mounted anywhere and certainly has many applications other than detecting broken glass!

Radio Shack also carries a surprising number of magnetically operated switches. They are intended to be used on doors and windows where the magnet is mounted on a moveable door or window and the switch is attached to the frame. Available in both N.O. (#49-512) and N.C. (#49-495) styles, when the matching magnet is in close proximity to the switch the switch is activated. Move the magnet away and the switch reverts to it's normal inactivated state.

Experimenters will see all sorts of uses for these. For example, low speed tachometers, pendulum clocks where the magnet is secured to the pendulum, and crude magnet strength measurement are just a few ideas. It is even possible to make a homemade anemometer constructed using the "egg" packaging cups from L'Eggs pantyhose. These cups can be attached to the ends of two small pieces of wood mounted at right angles around a bearing. A magnet can then be mounted on two of the pieces of wood so that the matching switch is activated every half turn of the anemometer. With calibration, measuring wind speed becomes a matter of measuring the elapsed time between magnetic switch closures.

Foiled Again

Even the adhesive foil used to detect glass breakage (#49-502) can be considered to be a switch. It obviously falls in the normally closed category and conducts electricity until a crack in the glass breaks the foil and therefore the circuit. There are also 2 heat sensors built to detect fire that are also usable as switches. Catalog #49-482 will activate at 135 degrees and #49-483 trips at 190 degrees.

The Radio Shack catalog contains several key operated switches as well. Cat. #49-523 is the momentary type while #49511 has locking contacts, and #49515 has a spring loaded cover. These are normal switches, except that turning them on or off requires a key. One takes a normal key, the other a circular one. There is even a programmable digital key switch (#49-535) where after entering the correct number sequence on the calculator-like keypad, the normally open switch closes for a short time. Obviously, this device was designed for use with a burglar alarm and it requires a separate power supply, but nevertheless, it is a type of switch with unique possibilities when used with an Apple][.

Using Switches with the Paddle Inputs

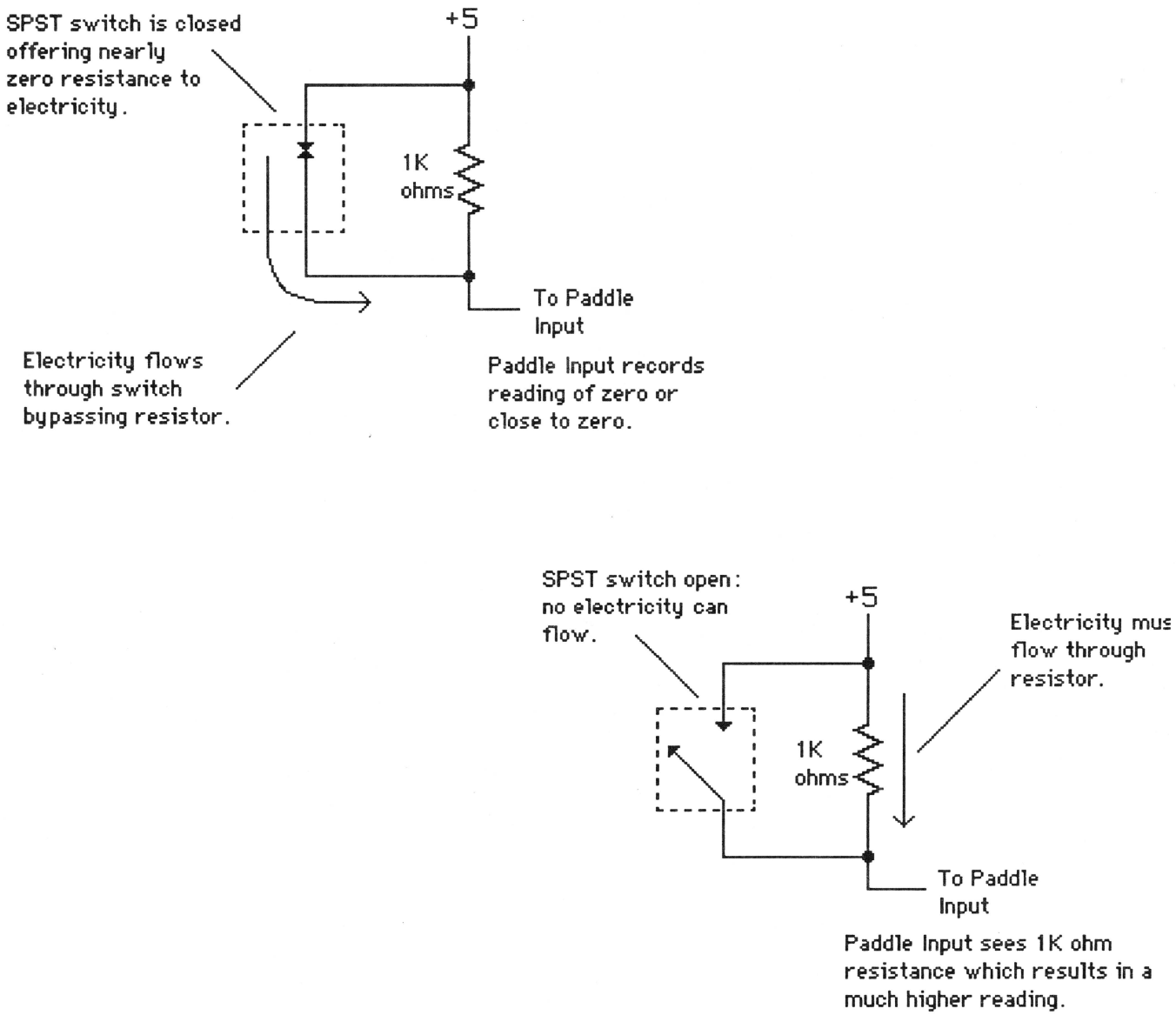
There is a way to use switches with even the paddle inputs. You will recall that the paddle inputs measure a resistance - usually a variable resistor in the joystick or paddle. By using switches and fixed resistors of known values in parallel, it is possible to use the game port to measure which switch (or switches) are open. This is a clever idea borrowed from several of Forrest M. Mims electronic circuit idea books.

It works like this: electricity always follows the path of least resistance. If you place a closed switch and resistor in parallel the electricity will always flow through the closed switch. (See Fig. 3). Because of this, the paddle input measures little or no resistance which shows up as a very low reading. If you open the switch, however, the electricity has no choice but to flow through the resistor. This results in a different (higher) paddle reading. Several switch/resistor pairs can be used in a circuit to form a network with all sorts of possibilities.

By carefully choosing the values of the resistors in a multi-switch system, you can determine not only if a switch is open, but which ones and how many are open at a time. For example, if the resistor at switch one yields a paddle reading of 23 and the resistor at switch 2 gives a reading of 106, knowing which switch is open becomes a simple matter of reading the resistance of the circuit. In addition, if the paddle reading is more than 106, then you know that both switches are open at once! See Fig. 2.

Carrying this idea even farther, by choosing resistors with unique values that compliment one another, it is possible to make a network of many switch/resistor pairs. Suppose you used the

Figure 3 - Using Switches with Paddle Inputs



following resistor values all with a 5% tolerance:

Switch 1: 1K	Switch 4: 10K
Switch 2: 2.2K	Switch 5: 22K
Switch 3: 4.7K	Switch 6: 47K

No matter which switches were open when, you'd get a unique reading for each combination. For example if switches 2 and 6 were open, the resistors would add making a total resistance of about 49K. The paddle reading for this resistance

would also be unique.

There are a few problems with this scheme. The resistors listed above are off-the-shelf sizes and attempting to get many different values where all the combinations add to unique totals less than 150K ohms is difficult. If you added a seventh switch, the resistor would need a value of 100K ohms which would cause problems. Obviously, if you open switches 6 and 7 at the same time, both resistances would add (47K + 100K= 147K) placing you near the edge of the range for the paddle input (150K ohms). Opening both switches gives a paddle reading of 255 so opening additional switches has no effect. These problems would be eliminated if you had access to precision resistors with the values associated with normal binary place value weighting such as: 1K, 2K, 4K, 8K, 16K, 32K, and 64K ohms.

This idea has applications in all sorts of areas. Like we discussed before, most burglar alarms use a loop of normally closed switches. Obviously, when the circuit is broken, the alarm system is activated, but the question remains, which switch broke the circuit? An easy solution is to design the system using the switch/resistor pairs so that the open sensor switch can be identified by just measuring the resistance of the circuit.

Four (or more) magnetic switch/resistor pairs could be mounted on a weather vane and the magnet attached to the vane itself to form a wind direction transducer. Wind direction could be determined by reading the paddle which measures the resistance of the circuit. Although admittedly crude, this device plus the anemometer discussed above might form the beginnings of an Apple-based weather station.

A Hardware Key

A hardware key is a device that must be connected to your computer in order for some software to work. Basically, all that happens is the software checks for the presence of the hardware key. If the key is present and responds properly to the software, all is well. If the key is absent, or not working properly, then the software aborts and the user is denied access. In the IBM world, hardware keys are often connected to a parallel printer port. It is a relatively expensive, but fairly successful way of controlling access to a program or data.

We can use the switch/resistor pair idea to make a hardware key. Follow the schematic in Fig. 5. I built mine on a small piece of board (half of #276-148) using the 8 position SPST DIP switch (#275-1301). If you're careful you can solder the resistors between the terminals for each switch then cut away most for the board. This small package can be mounted on the inside of the computer with double faced tape where it won't be seen. (If someone doesn't even know the key exists, it will be harder to "crack".)

Another idea is to mount the circuit board inside a small box (#270-230) so that it cannot be seen at all. Only the wires and connector to the game port would come out of the box. To use the key, just plug it into the game port of whatever computer you like. This makes for a portable unit. The drawback is that to change the switch combination, you must open the box. If you wanted to get really fancy, you could mount a gameport connector on the box as well so you can connect both the key and a joystick to one gameport at the same time.

If you have any Apple except a //c or //c+, you can connect it to one of the very rarely used paddles - paddle input 2 or 3. I used a heavy duty chip socket (not sold by Radio Shack) where the pins are large enough to make contact when inserted into the 16 pin DIP game port connector. By carefully looping the 2 connection wires around the pins on the underside of the socket and using a minimum of solder, you can then insert it into the game port and plug a regular joystick into this assembly piggyback style.

This method makes the key transparent hardware-wise, since none of the normally used Apple features are disabled or interfered with, and also software-wise since paddles 2 and 3 are used only on rare occasions. As a result, all your hardware and software should function normally even with the key installed 100% of the time.

To check for the key, just read the paddle it's connected to. If the value returned is the value you expect, the key is present and operating. If not, then the key is absent or has the wrong combination and your software can then take whatever action is appropriate.

Not all combinations of switch settings are desirable. Note that switch 0 has no resistor. It acts like an on-off switch. If it is off, it effectively disconnects the key from your computer. This is useful if you want to use game paddles 2 or 3 for something else. When nothing is connected to a paddle when it is read, the PDL routine returns a value of 255. As such, using any combination of switches where switch 0 is off will result in a value of 255. This doesn't provide much security because the paddle reading is the same as if the key weren't there (255). In this case you could

disconnect the key and still gain access to whatever you're trying to protect!

If you set both switches 7 and 8 off in your combination, the paddle reading may also return 255 since those two resistors together add to nearly 150K ohms, the resistance that returns a paddle value of 255. The best solution is to use either switch 7 or switch 8 (or neither), but don't set both to "off" in any one combination. The general rule is this: use any combination of switches you like, just be sure they don't return a paddle value of 255 since this doesn't provide any security.

Using the Hardware Key

To use the key, flip the switches to any combination you like (and write it down!) then run this single line of Applesoft code:

```
PRINT PDL(X)
```

where X is the number of the paddle you are using. Write down the value you get as well. Each switch combination will yield a different paddle reading. Sometimes the hardware will alternate between 2 different paddle readings (like 161 and 162, for example) for a given combination of the switches on the key. To detect this, run the one line program 8 or 10 times. (It is best not to write a loop in Applesoft to test for this because reading the paddle quickly causes it to yield slightly different readings.) If the results are steady, then fine. If the values alternate, note what they are.

Once the combination is set, the key is ready to use. To protect your software or prevent unauthorized access to your data, you'll need to read the paddle that the key is connected to at one point. When you check for the key is up to you. Typical times are during boot up when you can deny access entirely without the key, or the first time someone tries to read or write to a disk. If you like, you can allow reading, but disallow writing to disk if the key is absent. This would protect someone from altering your data. You could also allow writing but prevent reading. This way you could have someone enter data, but not read any off disk.

Checking for the key in your own program is about as simple as you can get. You can do it by issuing a command like:

```
10 KEY=PDL(X)
```

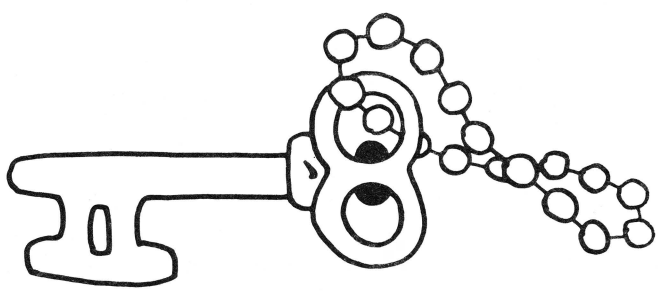
If key's value equals the value you got when you set the combination, then all is fine. You may have to allow for two correct values if the readings jitter back and forth for a particular combination of switch settings.

Neat Stuff

I ran across a neat publication that hardware hackers might be interested in. It's a magazine called "Electronics Handbook" and although some of you already know about it, many might have missed it since it is published quarterly. It's aimed at people who like to build projects and gadgets at the beginning or intermediate level.

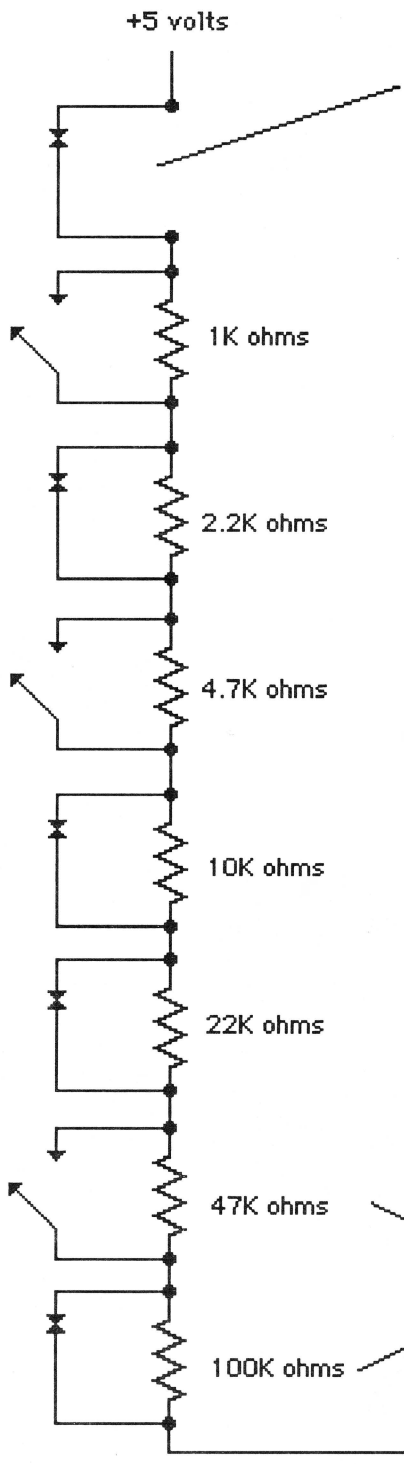
It's got book reviews of interest to hardware hacking types, an ongoing tutorial on different components (this issue: FET's), construction techniques such as how to solder, an annotated catalog listing showing what various electronics manufacturers offer, and lots of circuits, many with only 1 or 2 transistors or IC's. The issue I read even had an article on why your TV starts with VHF channel 2 instead of 1. In short, many of the ideas, projects, and circuits could easily be adapted for use with your Apple. Subscriptions are $12.00 per year (4 issues). Contact: Electronics Handbook, P.O. Box #5148, North Branch, N.J. 08876.

Want other hardware ideas? Check out a book on robotics. These days robots and computers go hand-in-hand. One current title is "The Robot Builder's Bonanza - 99 Inexpensive Robotics Projects" by Gordon Mccomb published by TAB Books (#2800). This 326 page book features many photographs, diagrams, circuits, and illustrations in 33 different chapters. Chapter topics include the use of stepper motors, robotic arms, circuits to imitate the sense of touch and sight, speech synthesis, smoke and heat detection, and many others. Almost all of these ideas and many of the circuits could be connected to your Apple with little or no adaptation. With a little imagination you could have your Apple controlling robot arms and all sorts of electromechanical things.



A Simple Security System

Although there are 255 unique settings of the 8 switches, only one pattern of switch settings will provide the "correct" resistance.



Note that there is no resistor here. This switch acts like an on/off switch. When it is open, the paddle input will return a value of 255.

On bootup, a "HELLO" program can chec the paddle input port for the correct resistance. If the paddle reading matches with the expected reading, the program proceeds normally. If not, access can be denied.

The circuitry for this "lock" can be plac very small circuit board and stuck on th of your Apple's case with double faced t

Electrical connection can be made using duty 16 pin DIP socket bywrapping the 2 around their respective pins and pluggin the game port piggyback style. A norma can be plugged into the piggyback socket electrical or physical interference.

Using these 2 switches toge but note that they total 147l maximum the gameport hanc It is best to use only one of switches at a time when dec "combination" of your lock.

Classifieds

In the early going, we had "Hired Guns", classified ads for programmers looking for projects. This went over well enough that we are opening up the service to *all* kinds of ads. Hardware, software, services, programming, etc. For an indefinite introductory period, we're still offering this service for free. Freebies are limited to 10 lines at 70 characters per line. You'll pay a dollar per line thereafter.

Classifications:

001	-	Hardware, Software For Sale
002	-	Help Wanted
003	-	Work Wanted (formerly Hired Guns)
004	-	Miscellaneous

There are a couple caveats:

• no X-rated BBS ads permitted

• we reserve the right to cancel an ad at any time and for any reason, including but not limited to space considerations.

We *welcome* commercial use of this space. If you've got a program to sell, you should list it here.

001 - Hardware/Software For Sale

If interested call Scott Scheuerman at (607) 336-5850. Make me an offer on any title or card. All can be sold separatly or in dundles. Most software titles come with full documentation but some have been lost over time. All are originals. I have no set prices so make any offer. I might take it.

Hardware for sale:

1) Generic 80 column card II+
2) PCPI AppliCard with 192K 6 mhz II+, IIe
3) Mocking Board speech and sound card, II+, IIe
4) Apple Dumpling Parallel printer interface II+, IIe
5) Crackshot copy card II+
6) Apple SCSI Rev C Interface II+, IIe, IIgs
7) Cirtech 512K Battery Backup RAM card II+, IIe, IIgs any slot no wires draws power from slot. Great for BBS's or booting system software 100ns RAMs. Software, Inc.
8) EOIC 2400 Classic Internal modem

Software for sale:

Paintworks Plus GS
TML Pascal I GS with source code and speech lib.
LPA Micro Prolog
Visible Computer 6502 Assembly Teacher
Turbo Pascal Version 3.0 CP/M

FAST EXTENDED PRECISION APPLESOFT
Choose 12, 14, 16, or 19 digits precision. Modifies + - * / ^, SQR, LOG, EXP, SIN, COS, TAN, ATN. Runs at assembly language speed on 64K Apple II+ and IIe/IIc/IIgs. No compiling - modifies Applesoft and supports the same comands. Test version available for comments. $25.00 ppd on fast-load DOS 3.3 5.25" disk. RFI Software, 9719 Crystal Lake Drive, Woodinville, WA 98072

---

003 - Programming Work Wanted

Bryan Pietrzak. 202 E John, #A16. Champaign, Il. 61820. GEnie: [BRYAN.ZAK]. America Online: [Bryan Zak]. Can do Pascal or 65816. I'll only work with 16-bit stuff. I like databases, GS/OS, NDAs, CDAs, desktop apps, text screen stuff. You name it.

Douglas Gum / O.P.Software, P.O. Box 1042 Mahomet, IL 61853 (217) 586-2904 I work mostly in 8-bit assembler,. Have done Awks enhancements, graphics/animation, and loads of custom patches for making 'incompatable' programs work anyway.

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).

Chris McKinsey, 3401 Alder Drive, Tacoma, WA, 98439, 206-588-7985, GEnie: C.MCKINSEY. Experience in programming 16-bit (65c816) games. Strengths include complex super hi-res animation, sound work (digitized and sequenced), and firmware. Looking for new IIgs game to develop or t0 port games from other computers to the IIgs.

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

Ross Indulges in Ironic Hacking

rStringList in Theory and Practice

by Ross W. Lambert

Why in the world would I want to tackle something as off -the-wall and unexciting as the string list resource? Well, there are three reasons, really. The first is that it is an easy resource format to understand. The second is that it is an easy resource to create. Third, it is a wonderful tool, useful in many situations.

Convinced?

Nah, me neither. So I'll prove each point as we go...

Wha Issit?

A string list is exactly what it sounds like: a list of strings. The first word in a given string list resource is a count of the number of strings contained therein. Hence you can have up to 65535 of 'em. What follows thereafter is simply a bunch of Pascal strings, that is, length byte/string/length byte/string, ad nauseum up to the number of strings in that resource.

Easy.

They are useful little buggers in a variety of situations, the most obvious application being any time you need to keep track of a list of strings (yeah, I know, "Duh".)

For example, let's say that you wrote a spelling game for educators. You could (and probably should) include an option wherein the teacher can add their own words. You could store the word list in a separate text file, but there is a good chance the words and your program could get separated. Then you'd have to ask where the data file lived, etc. This whole song and dance is tacky when avoidable. And it is avoidable.

If, instead, you stored the spelling words in a string list resource, they'd always be handy. Heck, if there was room and their attribute bits were properly set, they'd even be in RAM.

It don't get better'n that.

Other Uses

Now if you want weirder applications of the beasties, I got 'em. I like to muck around with PostScript, the page description language gurgling around inside your favorite LaserWriter IINT or better. Since PostScript programs are merely lists of commands (in ASCII no less), string lists are absolutely ideal. I can have one utility that does a jillion things. It just sends a different string list to the printer depending on what I want to do.

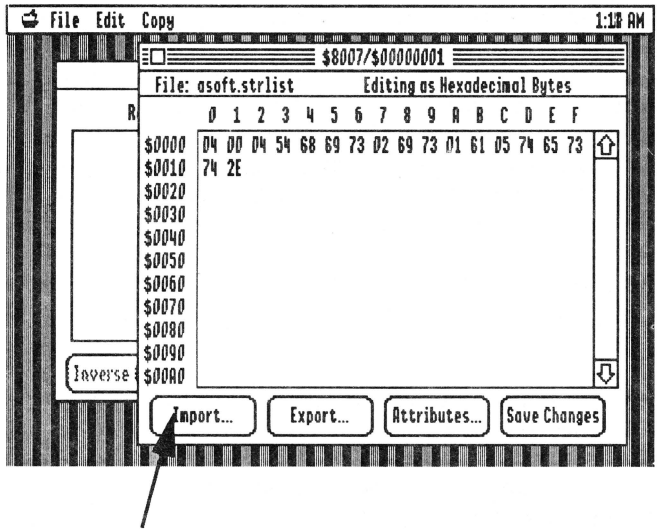
And if you're into copy protection, you could "sign up" the user when the program is first run, saving their name and/or serial number into a string list resource. From then on the startup screen would say "Licensed to Joe Smith" or whatever. This is big time stuff on the Mac. All the really expensive applications do it even though it stops absolutely nobody. The copy of Microsoft Word that was on my ex-superintendent's hard drive used to announce that it was licensed to Kinko's Copy Center.

There's probably less annoying and more worthwhile uses of string lists than psilly pseudo-copy protection. But since many of you will find yourself doing work for more annoying and less worthwhile project managers, I thought I'd toss it out.

Okay, How Do I Get One?

At the present state of the tools available, you have

Figure 1 - LLRE's Hex Editor



The import button: allows you to suck up a data fork into your current resource.

two choices for creating a string list resource. Those of you with APW/Orca and Rez can and should use Rez. I'll show you how to do that in a moment.

The rest of you should use LLRE (Jason Coleman's Low Level Resource Editor - shareware on GEnie or on our September disk) and Applesoft.[1] Hehehe. I'm not kidding.

LLRE has one *really* neat feature - you can import a data fork into a resource fork. That means that you can format data using whatever tools you have at your disposal - word processors, the monitor, Applesoft, whatever - and LLRE will deposit it into the resource of your choice.

I found the easiest scam for creating a data file in the string list format was to use Applesoft. Here's the listing...

Listing 1 - StringList Data Creator

```
10 D$ =  CHR$ (4)
20 S = 16384
40  READ NS : REM get number of strings
50  POKE S,NS -  INT (NS / 256) * 256
60  POKE S + 1, INT (NS / 256)
70 S = S + 2
80  FOR X = 1 TO NS
90  READ T$:LT =  LEN (T$) :REM get str
100  POKE S,LT : REM POKE length byte
120  FOR C = 1 TO LT : REM POKE Chars
130  POKE S + C, ASC ( MID$ (T$,C,1))
140  NEXT
145 S = S + LT + 1 : REM inc byte count
150  NEXT
160  PRINT D$;"BSAVE STR.DATA,A16384,L";S-
16384
199  END
200  REM
201  REM  Change these to suit
202  REM
210  DATA  4 : REM number of strings
220  DATA  "This","is","a","test."
```

This quick and very dirty little Applesoft program does nothing but create a block of memory that looks like string list resource and BSAVE it. The rest is up to LLRE.

The first thing to do in LLRE is to create a new resource file. Having done that, open it and select the NEW button.

You will then be prompted for the resource type (by number) and ID. The number for the rStringList is $8007. The ID is whatever you want it to be. Make it easy to remember since you'll have to use the same ID number when developing your source code.

At this point you will be dumped into the hex editor (see Figure 1), where you want to select the Import button. Find your data file (I named it STR.DATA in the Applesoft program. You can name it whatever suits ya.), select it, save the resource, and you're done.

If you compile or assemble an application that uses this string list, you probably ought to use a different file name and copy the resource fork to the application via Rez or LLRE. This way you won't accidentally overwrite your new resource.

This is not strictly necessary with Merlin 8/16+. My version, at least, only re-creates the data fork when you do a new assembly, thus your resource fork is preserved between versions of the application. This is a very nice touch. Thank you Glen Bredon, wherever you are.

Rez

The procedure in Rez is simplicity itself. It's not as

1. Matt: yes, yes, yes, I know. Rez does everything and isn't it really stupid to use Applesoft. I agree wholeheartedly - for those who have Rez and APW/Orca. But for those who don't, isn't it silly to spend $180 (suggested retail total) to do one of the very few things that Genesys cannot yet do? I can jump through a hoop for $180.

much fun as doing the totally ironic and bizarre (i.e. an Applesoft resource developer!), but we can't have everything.

Here's the Rez code to generate the string list resource:

Listing 2 - Rez Code for rStringList Resource

```
/* Sample Simple Rez Stuff */

/* So you can access predefined goodies... */
#include "Types.rez"
resource rStringList (001) {
    {   "this",
        "is",
        "a",
        "test"
    }
};
```

I just type "compile rez.source keep=outputfile" at the command line to actually create the resource.

Where did it go, George?

The most common complaint I hear about resources is that they're harder to use (and find) than data embedded in source code. Even if that were true, their benefits outweigh the disadvantages. But for the sake of simplicity, I have created a macro (GetResStr) and subroutine that will make string list resources just as easy to use as embedded strings. If you call the macro with just a little information handy, it will return a pointer to the string so you can work with it just like normal.

There is one really big GOTCHA, however.

If you expect your string to stay where GetResStr says it is, **the resource must be locked down**. The routine locks it down on its own (for safety sake), but it does not unlock it. It couldn't really, since it has now way of knowing when you're through with the data. (By the way: When you create a resource, if you set the resource attributes bits such that the resource is locked when loaded, you don't have to lock them down from your program. That being the case, you could remove the HLock call in the DigOutStr subroutine.)

Keep in mind, however, that there is no reason to keep a resource locked in place if you're not going to use it again, so some judicious memory management is in order. Too many locked blocks can fragment memory (A Bad Thing).

For more info on locking and unlocking blocks, check into chapter 12 of volume 1 of *The Apple IIgs Toolbox Reference*. A block of memory that happens to be a resource is not a whole lot differerent than any other block - as far as the Memory Manager is concerned. But there are two very important caveats if you dink with things on your own (as opposed to setting the attribute bits at the time of creation): **1) do not, I repeat do not, dispose of a block (i.e. handle) that the Resource Manager has allocated,** and 2) unlock the handle immediately after you're through with it (presumeably after printing the string, for example).

The latter is just due to my gut feelings, the former is an absolute, chiseled in concrete, Thus Sayeth Apple dictum.

The following code is called GetResStr because you pass the macro the ID of the resource you're after (That's *your* ID, boys and girls. GetResStr assumes we're working with rStringLists, which is the resource *type*.) the number of the string you're after (the first in the list is 1, etc.), and the address where you'd like the address of your string deposited. This last parameter could be the address of (i.e. a pointer to) a variable if your primary environment is a higher level language.

Please NOTE: This is not a complete program. I'll have a complete demo on the monthly disk, but space constrains me from including the entire thing here. All you have to do to create a your own demo is to throw in startup, shutdown, and window routines. Open a window and draw a few strings from the string list and you've got it.

Honest.

The bit twiddlers amongst you will probably wonder why I set up the temporary data area and pushed and pulled values off the stack (using the Pascal protocol).

Well, for one thing it makes the subroutine self-contained. That is, you could rip it out and put it into a special subroutine module external to the main body of the program. With careful planning, you can have a generic subroutine module that never needs to be reassembled, just linked into everything you write. That is why Tempdata, TempInt, etc. are there - they are special generic data locations used by all of my subroutines. Saves a few bytes, anyway.

The syntax for a call to GetResStr is pretty easy:

```
GetResStr ResID;StrNumber;Ptr
```

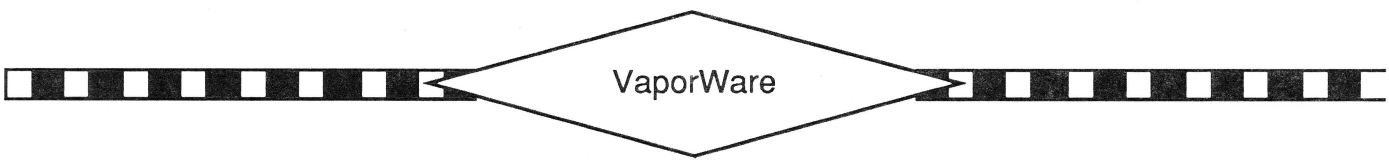
ResID, you'll recall, is a long word, string number is an integer (two bytes), and Ptr is the address where you'd like the address of the string deposited.

Miscellaneous Merlin Tip of the Week: Name your linker command file "Me" or "It". That way when you want to build your app it is just a quick "Link Me" from the command box. Believe me, that is much easier to type than "Link Snazzyness.cmds" and the like.

Listing 3 - Ross's ResStr Routines

```
*************************************************
*                                               *
*   FN GetResStr                                *
*                                               *
*   by Ross W. Lambert                          *
*   Copyright (C) 1990                          *
*   Ariel Publishing, Inc.                      *
*   Most Rights Reserved                        *
*                                               *
*   Merlin 8/16+ Assembler                      *
*                                               *
*************************************************
*

* NOTE: This is library code only - this is NOT
* an application that can be assembled.

* For resource stuff

MyResID  =     $00000001
rStringList =  $8007


***    macro definitions

* Check stack checks a long address on the stack
* for validity without disturbing it.

* The macro conditions the zero flag on exit.  A
* zero means the address on the stack is zero (an

* invalid pointer or handle), non-zero means
we're * okay.

CkStack  mac
         lda   1,s
         ora   3,s
         eom



* Syntax: GetResStr ResID;StrNumber;Ptr

* This routine sets the carry if the string
number * requested is out of range (i.e. there's
not that * many strings in the string list).

* The range of strings numbers you can request
* goes from 1 to n.

* GetResStr locks the string list down, but
does
* not unlock it!

GetResStr mac
         PushLong #]1      ;push the resource
ID
         PushWord #]2      ;and the # of str
wanted

         jsr   DigOutStr

         PullLong ]3       ;pull address into
ptr

         eom    ;check carry for
success/failure


.................


* Subroutines

DigOutStr
         PullWord RetAddr

         PullWord TempInt      ;number of
string
         PullLong TempID       ;our resource's
ID

         pha                    ;result space
         pha

;the type as Apple defined it
         PushWord #rStringList

;my resource's ID as I defined it
         PushLong TempID
         _LoadResource

         CkStack               ;neat macro
         beq   booboo

         lda   1,s   ;copy hndl to resrce w/o
         sta   TempHandle ;popping from stack
         lda   3,s
         sta   TempHandle+2

         _HLock                ;lock it down

         lda   $00  ;save direct page 00 - 03
         pha
         lda   $02
         pha

         deref TempHandle;TempPtr

         lda   TempPtr
```

```
        sta   0
        lda   TempPtr+2
        sta   2
        lda   [00]        ;this gets # of strings

        cmp   TempInt     ;compare to one we want

;if # of strings is equal or greater then okay
        bcs   okay

;set carry if out of range or resource error
booboo  sec
        jmp   exitfn

;offset - first string is after string count word
okay    ldy   #2
        sty   Tempdata
        ldx   #1          ;our counter

:loop   cpx   TempInt     ;is this string we
want?
        beq   :calcptr    ;yep
        clc
        lda   [00],y      ;get length byte
        and   #%00000000_11111111 ;mask off junk
        inc    ;bump length byte + length of str

;Tempdata holds 2 + length bytes of each string
        adc   Tempdata
        sta   Tempdata
        tay       ;slide over to y for new
offset
        inx       ;inc the count
        bra   :loop

:calcptr clc
        lda   Tempdata    ;offset of 64K or
less
        adc   TempPtr     ;add to ptr's low
word
        sta   TempPtr     ;and store it back
        lda   #$00        ;in case of a carry
        adc   TempPtr+2
        sta   TempPtr+2

        pla
        sta   $02         ;restore direct
page
        pla
        sta   $00
        clc    ;make sure carry is clear on
exit

;put pointer to string on stack

exitfn  PushLong TempPtr
        PushWord RetAddr ;point me home

        rts
.......................

* Temporary data for subroutines

Tempdata adrl  0
TempHandle adrl 0
TempID   adrl  0
TempPtr  adrl  0
 TempInt  dw    0
 RetAddr  adrl  0
```



Note: VaporWare is primarily for entertainment purposes. The views expressed within are those of the author and do not necessarily reflect the views of Ariel Publishing managment or editorial staff.

by Muphy Sewall, from the APPLE PULP, HUGE Apple Club (E. Hartford) News Letter

A Real 3-D Display: Texas Instruments has shown a "bubble" display two feet in diameter which "floats" three dimensional images within a volume. Multiple viewers can see the display from any side without special goggles or eyeshades. Dubbed "Omniview," TI's patent application describes the technology as a "real-time, auto-stereoscopic, multiplanar 3-D display system." Initial commercial applications may appear as early as next year. - *InfoWorld 20 August*

Motorola 68040 Delayed Again: Although volume production had been planned for last month, Motorola officials found a few last minute bugs (described as "very minute") to correct. Volume production of the chip for machines already introduced by Hewlett-Packard and NeXT and anticipated from Apple is now scheduled for the end of October or early in November. NeXT's planned October 15 shipping date seems likely to slip. - *InfoWorld 17 Sept.*

Intel i586 Design Note : Microsoft's William Gates and Intel's David House are discussing whether to build the graphics primitives of Windows 3.0 and OS/2's Presentation Manager into the mask of the forthcoming i586 chip. Such a decision would markedly improve the performance of both graphic user interfaces. - *PC Week 27 August*

Downward Compatibility: Microsoft's MS-DOS 5.0 will contain a Set Ver (set version) command that will allow users to make the operating system emulate earlier MS-DOS versions from 2.0 up for those applications that turn out to be incompatible with DOS 5.0. - *InfoWorld 20 August*

Even Larger Capacity Hard Drives: IBM is said to be planning to announce a 200 MByte magneto-optical drive with a $1,500 retail price. Only 10 years ago the 2 Mbyte hard drive introduced with the Morrow computer was viewed as large enough to meet anyone's storage needs. Now hard drives with capacities exceeding the Morrow's by over 1,000 times (2 gigabytes) are described as "just around the corner." - *InfoWorld 10 September and PC Week 27 August*

Another Source of Laser Printers: Compaq appears to be planning an aggressive entry into the laser printer business. The company will manufacture printers in Mexico that will print faster and cost less than the popular HP III. - *PC Week 27 August*

ClarisShare: Now that Apple has decided not to spin-off it's software division after all, the Claris label will begin appearing on important Apple software including HyperCard, AppleShare, and probably, the as yet unannounced AppleMail program. A Windows 3.0 version of Claris's FileMaker product is in the works, and Claris executive want to proliferate Apple technology onto other platforms and create "interoperable superworkgroup applications" (applications that could be shared among Macintosh, PC, and Unix workstations connected to a network). - *InfoWorld 27 August*

HyperCard 2.0 Delayed (Again): Apple tried and failed to get numerous Apple publications to delay for a month advertising that will appear in the November issues. The guess is that the ads (and stories) will be about HyperCard 2.0 which has been delayed until November, at least. The "final" beta version was sent to developers in late September. - *read on AppleLink 19 September*

dBase for UNIX: Now that Ashton-Tate has finally shipped dBase IV 1.1, the company has begun beta testing a version for UNIX. No definite release date or price has been set as yet.- *InfoWorld 20 August*

Intel's 50 MHz i486 Delayed.
Intel's microprocessor group president, David House, has admitted that plans to deliver a 50 MHz i486 CPU this year were "overzealous." Large PC manufacturers have been told that volume production will not begin until sometime in early 1991. Meanwhile, PC manufactures will also have to adapt to Intel's soon to be announced decision to streamline their product line. By the end of next summer, Intel plans to slim down to only three CPU's -- the 20 MHz 80386SX, the 25 MHz 80386, and the 33 MHz i486. - *PC Week 27 August and 10 September*

Executive Pen-Based Computer: Active Book Company will introduce a pen to glass input (with optional keyboard) computer next spring. The four pound notebook sized machine will cost about $2,000 and receive FAX and record voice mail as well as edit documents and search data bases. The CPU will be an Acorn RISC processor and the planned operating system is UNIX-based Helios, but licensing Go Corporation's technology (see last month's column) has not been ruled out. Active Book's computer also will come with an MS-DOS emulator and battery life of eight to ten hours is anticipated. - *InfoWorld 20 August*

Miniature Production Studio. Newtek Inc. of Topeka, Kansas will offer a $1,595 VLSI board named the "Video Toaster" for the Amiga Computer. When used with Newtek's point-and-click Light Wave software (bundled with the Toaster), the Amiga becomes a miniature production studio for less than $5,000 that can perform numerous editing functions at a professional level. Newtek's Toaster is a video switcher, effects generator, dual frame buffer, and character generator with a 16.8 million color, RT-170 resolution NTSC output. The largely intuitive New Wave software is accessible to users without specialized video training. - *InfoWorld 3 September*

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

Special Renewal Order Form

Place a check in each box that applies, add it all up, select your payment method, and **mail to Ariel Publishing, Inc., Box 398, Pateros, WA 98846. Or call (509) 923-2249 voice or (509) 689-3161 fax.** Don't worry about your name and address (too much). We've got it on your mailing label above!

☐ Sign me up for another insightful and enlightening year of *8/16* magazine (hard copy)..... $29.95

☐ Sign me up for two more insightful and enlightening years of *8/16* magazine (hard copy)..... $56.00

A: Total This Box
$_____

☐ Sign me up for another 8 megs or so of source code and utilities! I want one more year of *8/16 on Disk*..... $69.95

☐ I want nearly 16 megs of goodies. Give me *8/16 on Disk* for two more years..... $119.95

☐ I can't live without it even though I can't afford a full year. Give me 6 months of *8/16 on Disk* for $39.95

Circle Disk Size:
3.5" **5.25"**

B: Total This Box
$_____

☐ What a deal! Send me SSSi's DeskPak for **$15.00** and then also extend all of my subscriptions one month. This is okay to do even if you're not renewing!

☐ Since I ordered both *8/16 on Disk* and the hard copy of the magazine, I get to deduct $10!!! **Subtract $10 from total!**

C: Total This Box
$_____

GOTCHAS: If you live in Canada or Mexico, please add $5 per year per subscription (both disk and hard copy - sorry!) If you live outside North America, add $15 per year per subscription (disks go first class, the magazine goes Publisher's Periodical rate. First class magazine is possible for $45 per year in addition to the subscription price.) Washington State residents add 7.5% sales tax.

D: Total Gotchas
$_____

Method of payment (circle one):

Check Visa/MC Bill Me

If card, then #_____
Expiration Date_____

Signature_____

Grand Total (add A,B,C, and D):
$_____