A2-Central™

formerly Open-Apple

August 1991
Vol.7, No.7

ISSN 0885-4017
newstand price: $2.50
photocopy charge per page: $0.15

A journal and exchange of Apple II discoveries

Apple's booth shuns Apple II

Attendees at the National Educational Computing Conference in Phoenix, Arizona, report that Apple Computer's exhibit did not sport a single Apple II computer. Apple's Apple II team told developers such as Roger Wagner and Zip Technology that their cohorts in K-12 Education Marketing were responsible for the show and that they had been unable to persuade the education people to include Apple II systems.

This is one of the negative side effects of the consolidation of Apple's Apple II champions into the Apple II Business Unit. Having all the Apple II people inside Apple's new Consumer Products Division frees the other Apple divisions from having to maintain the pretense of support. Apparently Apple's head of education marketing, Bernard Gifford, who stood side by side with John Sculley last year promising continued support for the Apple II, has defined "support" in a way that doesn't include acknowledging that the Apple II continues to exist.

Refusing to exhibit Apple II computers at a show where over 60 per cent of the computers used in other booths were Apple IIs is beyond inexcusable. It shows that Apple's education strategy is to force the Macintosh into America's schools even at the risk of huge losses in market share, and more importantly, without regard to what it will cost the American education system. Apple could be solving the problems of today's budget-constrained educators by showing them how to add to and enhance the computers already in schools. The Apple IIgs does what a Macintosh does, but in color, and it also runs the software schools already own. But Apple's solution for education is to convince administrators that the choice for schools in the 1990s is MS-DOS or Macintosh. And taxpayers get saddled with new hardware, be it MS-DOS or Mac; new software, be it MS-DOS or Mac; and retraining millions of teachers — all because Apple's vision isn't big enough to include alternatives based on its own computers that are already in place in America's schools.

The good news is that more than 1,100 educators took the time to attend Roger Wagner's *HyperStudio* seminar at the conference. Roger told us that NECC also offered all-day workshops with the IIgs and these were well-attended; he expects the word about the intrinsic value of the IIgs as the multimedia machine of the 1990s to be passed along by these participants, if not by Apple itself.

Apple and IBM are rumored to be discussing a technology exchange. A copyrighted article in the June 15th *Wall Street Journal* asserts Apple and IBM may be considering the exchange of powerful IBM RISC (Reduced Instruction Set Computer) technology for Apple's expertise in designing object-oriented software, possibly representing an attempt to provide a technology alternative to Microsoft's MS-DOS/Windows environment. No announcement has been forthcoming as yet.—DJD

The People vs Programming

There have been stages in our history where the inability to adopt some skills has caused a type of class stratification; most of these stages followed a technological revolution where the ability to perform certain actions became a new requirement of daily life. Literacy became an increasingly important skill after the invention of the printing press. The Industrial Revolution altered the structure of society.

Our current society is undergoing such a change dealing with the mediums of recording, manipulating, and exchanging information. There have been two related references to this stratification in the past month; one a fictionalized accounting, one a real-world development.

One is a scene from the current movie *City Slickers*. Two men isolated from "civilization" on a cattle drive finally have the opportunity to discuss the vagaries of VCR use at length; Billy Crystal's character is explaining to Daniel Stern's alter ego why it isn't necessary to have the television tuned to channel 3 to record. When he explains the television doesn't even have to be *present* to record, the other man is aghast. A third character who has been listening to this conversation finally explodes, pointing out that some of the cattle no doubt have learned to program the VCR at this point, and that further instruction is never going to help Stern's character.

Serendipitously, there was a real-world fallout to this: that very month, a new device to "automate" VCR programming appeared on the market. The new device is an intelligent remote control called the *VCR Plus*, which can be configured for use with most common cable converters, televisions, and videocassette recorders using infrared remote controls. The additional feature of the *VCR Plus* is that when you find a program you want to record in the television (or cable) listings, you look for a multi-digit number next to the listing and enter it into the remote. The remote will store the number and, at the appropriate time, start your VCR and record the program. This saves you having to learn how to program your VCR to record a show.

My reaction to the scene in *City Slickers* is sympathetic to Crystal's

"I'M WAITING FOR MY AUTOEXEC FILE TO RUN, SO I'M GONNA GRAB A CUP OF COFFEE, MAYBE MAKE A SANDWICH, CHECK THE SPORTS PAGE, REGRIND THE BRAKEDRUMS ON MY TRUCK, BALANCE MY CHECKBOOK FOR THE PAST 12 YEARS, LEARN SWAHILI, ..."

character; certainly it isn't unreasonable to patiently try to explain the use of technology to someone if they are willing to learn and if you have the time and inclination. On a cattle drive, there isn't much else to do; of course, most of us have outside pressures that restrict our ability to offer the kind of instruction that Crystal's character tried to provide.

My reaction to the *VCR Plus* is a little more reactionary because I see it as a step backward. It's easy for me to remember, for example, that *L.A. Law* is broadcast from 9 to 10 PM Thursday nights on Channel 4 in the Kansas City area. Learning to program that into a VCR takes some effort; you have to learn how to enter the information and you have to be careful to check that the data is correct lest you record an hour from a "home shopping" station. It gets worse if you have multiple VCRs with different programming methods. A single device that would allow the consistent, logical entry of the information would be helpful.

But when you have to look up a number like "7067466," which does not correspond to anything in the way of human-interpretable data, have we actually *reduced* the complexity of our lives? You can at least remember the time slot for a program; there is no easily discernable rhyme or reason to the *VCR Plus* numbering system.

Meanwhile, our local *Kansas City Star* newspaper ***reduced*** the information content of the program listings because adding the 4 to 7 digit VCR *Plus* numbers after each program entry took up a significant amount of space in the listings. So the person who has expended the sweat to learn how to program their VCR is now penalized for the sake of those who won't. Is this the way technology is going to aid us?

There is even a phone number that you can call to find out the code for a program that isn't listed. All you have to do is provide the starting time, ending time, day of the week, and local channel for the program and an operator can provide you the number to enter for recording. What's amazing is that the individual is *still* programming the VCR; the difference is that the *VCR Plus* folks get to insert themselves profitably into the process by adding a layer that reduces the number of keystrokes you must enter, but totally obscures the meaning of the information you are entering.

It isn't that I think the *VCR Plus* concept is bad; if the same device allowed you a sensible, uniform entry of normal information (times, dates, channels, and so on) I think it would be a boon. My objection is that it is a use of technology in exactly the wrong direction; using a more obscure "code" to record makes you a slave to the *VCR Plus*. "Human readable" program listings in the newspaper are now trashed by additional redundant information and in the case of an absent listing you must call to find out the "secret code".

If you aren't willing to expend the effort to learn to program your VCR, you probably aren't going to thrive in a technological society. VCR manufacturers can try to make the task easier, but there is a point at which the process cannot be simplified without penalizing those who are willing to adapt.

We run into similar dilemmas as computer users. Many users today do not want to learn any programming to use their systems, but they also want the systems to act *exactly* as they wish. Cost can also be a factor; generally if you pay enough money, you can have most of your first two wishes, even if you have to hire someone else to compile your dream system from scratch.

Apple's philosophy of personal computers is based on a model of interdependency. As an example, Apple manufacturers computers, system software, and printers that are relatively tightly bound to each other so you can have programs like *GraphicWriter III* (for the IIgs), and *XPress* (for the Macintosh) that allow you to design and print documents of varying complexity without having to enter tedious details about your printer.

The drawback is that the printer you want to use has to be supported either by the specific software package (on the Apple II) or the operating system (for the Mac or native IIgs programs, where printer drivers are part of the system software); if you can't select the printer you want from the "pre-approved" list or if the implementation of the driver is not "complete" in your eyes, you're out of luck (unless you have the time and skill to implement and install your own driver).

Tom Weishaar dealt with the specific issue of printers in "Special issue: Solving printer problems" in the November 1985 ***Open-Apple*** (***A2-Central***'s old name). The major quote from that article remains true:

The bad news is that the only person who is in a position to solve YOUR printer problems is YOU. You are the one who selected the particular combination of hardware and software you have — there may be fewer than a hundred other systems exactly like yours in the world.

You are ultimately limited by what a system's hardware and software will let you do. Actually, in the real world where finances and time are involved, you may be limited by what a system's hardware and software will let you do cheaply and easily.

There are ways that you can get more control over your system. Most involve an education curve and only you can decide whether any improvement is worth the investment of your time to implement it. If you don't have the time or inclination, you may have to spend money, and if you don't want to invest the time, effort, or money then it's time to contemplate whether the computer was actually a good investment for you. Most of these decisions need to be made before you invest in a computer and re-evaluated when you add software or hardware. These days we're usually involved with people in the re-evaluation stages.

Hardware is easy to change, and sometimes that is the cheapest solution. Saving $100 on a printer is no bargain if it takes you 20 hours to get it working and your time is worth over $5 an hour. Similarly, it isn't exactly reasonable to buy a lower-cost printer and assume that someone else will get it working for you. The same goes for most other hardware items; follow the proverb that says "pay the money and cry once". Something isn't cheap if it doesn't work or requires constant tweaking.

Software is generally harder to change because it is even more labor-intensive to alter. As the demands upon the hardware increase the software also gets larger and more complex; most people don't sit down and write their own software from scratch these days (as they might have ten years ago) for that reason. Modifying a program that has been written by someone else can be even more difficult; you first have to figure out how the original program works, then see if it is feasible to add code to modify the program. In many cases, modification isn't realistic. (Many people who wouldn't dream of asking someone to modify printer hardware seem to think modifying programs is somehow "easier"; in most cases, it isn't.)

Fortunately, some software is designed to allow user customization and even control. But we're back to the VCR problem: if you want the software to do what you want, you have to be able to tell it what you want. That means learning to program, whether or not you like the idea.

There seem to be many users interested in learning how to program their systems, if the "guru" factor can be removed. We are moving into an era where the common programming languages are application-dependent macros, scripts, or even complete new languages like HyperCard's HyperTalk. With application-specific languages proliferating, requests for programming assistance are becoming as varied as printer requests, and it is as difficult for one person to know all the answers as it is with printers.

To be "universal", the focus on learning to program has to be on the elements of design that are common to *all* languages. Users have to adapt a problem-solving mentality and the associated skills. The job of translating the ideas into a specific language is then reduced to a more mechanical procedure.

The computer language itself is not the key to the solution. If you hold distaste for programming, first forget the images of High Priests spouting computerese gibberish in COBOL or FORTRAN that are usually associated with the word. A program is only a series of discrete steps applied to produce a specific effect. Those steps can be described in your spoken language — English, French, German, Italian, or whatever. As a matter of fact, it's preferential to write the program description in your spoken language before you start doing anything else.

If you have trepidation about programming, this realization should remove it. You do this type of problem-solving every day. To enter your house, you may need to insert a key, turn it, turn it back, remove it, grasp the doorknob, turn it, push the door open, and release it. Your specific program may vary, and you normally don't stop to analyze the steps, but the point is you can do this. *You do it every day.*

Of course, there is a certain drudgery in breaking everything down into "baby steps", and unfortunately computers require you to be very explicit since they aren't very intelligent. A computer can execute instructions very quickly, but it only understands what it is told, and it has a limited vocabulary. When the computer reaches an instruction it doesn't understand, it will either misinterpret it (venturing into some alternative path until it can no longer proceed) or will simply be unable to do anything and stop. This is one way "computer errors" (actually errors from the human writing the instructions) occur.

If you want control over your system, you will have to decide to deal with the drudgery. Think of the *VCR Plus*; if you can't (or won't) make the effort to solve the puzzle, you will have to live with what others hand you.

Programming in your native tounge. The first step in designing a program, whether it is in a "traditional" computer language or an application-specific language, is to sit down and analyze how the task you want to automate is performed. That is, how would you explain the steps involved in the operation to a person who had never performed the task before?

Imagine the instructions to average three numbers as provided by someone else. The person doing the averaging has to get each of the three numbers to average, add them together, then report the average. But assuming they don't know what "average" means, this may have to be refined further stepwise:

```
1) Get and record the first number.
2) Get and record the second number.
3) Get and record the third number.
4) Add the first and second number and record the result as the first sum.
5) Add the third number to the first sum and record it as the second sum.
6) Divide the second sum by three and record as the final answer.
7) Report the final answer.
```

The tedium starts to show as we move further toward the realm where we assume **nothing** about the entity performing the averaging. Well, actually we still have to assume some things, such as their ability to add and divide.

Once you have the problem explained in natural language, you come to the book work: learning enough about a programming language to translate the actions. You must understand how the actions are phrased in the computer language. And you must also know what won't translate directly and therefore must be further simplified or redefined.

In terms of computers, the programming environment you write for will have varying degrees of knowledge. At the lowest level, a computer only understands a very limited number of instructions that perform very limited actions. For example, the 6502 microprocessor in the Apple II works primarily with 8-bit values (numerical values between 0 and 255) and is limited to moving these values around in the computer's memory, addition, subtraction, comparisons, and a few other operations. Everything else, *even multiplication and division*, must be derived by programs constructed from these basic operations.

The level also affects how many details we must know about how the computer itself works. The lower the level of the computer language, the more the programmer must know about the computer itself. That's why machine language (the "native" language of the computer) is the most tedious of all. Most of us strive to use something with more powerful commands.

As we move up the programming language scale and isolate ourselves further from the "guts" of the computer, our environment becomes more sophisticated in terms of its command vocabulary but may become more restrictive in what can be accomplished. For example, writing a disk operating system (which has to communicate directly with the disk hardware) in Applesoft is not feasible. Having the problem carefully defined in a language we understand helps us determine which computer language we should use to solve it.

Users seem to have different levels of intimidation. Asking a user to write a BASIC program to solve a problem may result in frustration. Yet we've run into very few users who feel intimidated by writing a formula for a spreadsheet. It is key to recognize that the act of programming is generally the same and the differences lie mainly in the syntax and relative power of the target "language".

What we'd like to do is look at the progression from a "classic" language like Applesoft to application-specific languages. This is not meant to teach any of these languages in a few paragraphs; normally you would learn the language from books or possibly in a classroom. What we want to get across is that programming should not be intimidating and that languages are progressing toward a form that we believe will encourage their use.

The translation to a "real" programming language. Assuming we're content to take the shortest route to a working solution for our averaging problem, the conversion to Applesoft is straightforward from our stepwise description. The overall problem is still the same: allow the user to enter three numbers, calculate the average, and display it. One way of solving this in Applesoft is a near-literal translation:

```
10 TEXT
15 HOME
20 INPUT "First: ";N1
30 INPUT "Second: ";N2
40 INPUT "Third: ";N3
50 S1 = N1 + N2
60 S2 = S1 + N3
70 A1 = S2/3
80 PRINT "The average is ";A1;"."
```

There are some differences from our English program. In Applesoft, we have to have a distinct identifying number for each instruction line; lines in the program are executed in the order of these numbers unless we use an Applesoft instruction (not among those used above) that causes a line to be executed "out of order". Also, although the words TEXT, HOME, INPUT, and PRINT have meaning to us, they have narrower, specific meanings to Applesoft. As in English, these language protocols are referred to as *syntax*, and the syntax of a program has to be correct for a computer to run it. *Absolutely* correct, because the computer is very bad at reading between the lines.

If you are unfamiliar with Applesoft, why and how the above program works may be a mystery. But even if you're intimidated by programming, it should also be apparent that we haven't moved very far away from our original description. What we have is a problem in *translation*, not some mysterious black magic. If you have the logical problems ironed out in your native-language outline and make the translation accurately, the program should run and produce the correct results.

The first line (TEXT) puts the screen in text mode, the second (HOME) clears the screen and puts the cursor in the upper left-hand corner; you might think of the combination as getting a blank sheet of paper to work on. INPUT allows us to print a prompt (the text within quote marks) and receive an answer typed from the keyboard into a variable (the item after the semicolon in each line). The equations in lines 50-70 perform the math, and line 80 displays the answer on the screen.

To execute the program, we use the command word RUN. Issued while our program is in memory, RUN causes Applesoft to clear the values of any variables and start executing our program at the first line. When prompted, we enter our three numbers, and Applesoft prints the answer.

Because Applesoft is sufficiently sophisticated to handle the math in a single equation (humans usually add only two numbers at a time), we are able to combine the calculations in a single step:

```
10 TEXT : HOME
20 INPUT "First: ";N1
30 INPUT "Second: ";N2
40 INPUT "Third: ";N3
50 PRINT "The average is "; (N1+N2+N3)/3;"."
```

If we wanted to try to duplicate the appearance and function of an AppleWorks spreadsheet the program would obviously be much more complex, but it is possible to write Applesoft programs that aspire to such imitative sophistication. (For an example of an Applesoft program that duplicates the "look and feel" of the AppleWorks database display, see "Reading AppleWorks data bases", March 1987)

Redefining the solution in an AppleWorks environment. Unlike Applesoft, AppleWorks's spreadsheet module provides the screen formatting and entry routines for us; PRINT and INPUT statements disappear. The penalty is that we have to start AppleWorks (a rather large program compared to Applesoft) in order to use the program.

Assuming that we don't want the three numbers to be part of a permanent formula, if we were to suggest a method of solution it would probably be to "enter each of the three numbers into a cell, then create a fourth cell who's value is the average of those three numbers".

Laid out in a spreadsheet (with column and row labels), the solution might look like this:

	A	B
1	First	1.2
2	Second	2.4
3	Third	4.8
4		==
5		2.8

The formula for the "Averages" cell (B5) could be done in a few different ways, but we'll choose to use the @AVG function:

```
@AVG(B1...B3)
```

If there were a programming language for the AppleWorks spreadsheet, the program to create this spreadsheet from the main menu might look like this:

```
type "1"
type return
type "5"
type return
type "1"
type return
type "Sample.SS"
type return
type "First"
type down arrow
type "Second"
type down arrow
type "Third"
type down arrow
type right arrow
type quote
type "===="
type down arrow
type "@AVG(B1...B3)"
type return
```

All of these instructions involve entering data at the keyboard. We use "type" followed by the name of the character ("return", "quote", "down arrow") or text in quote marks to indicate what needs to be typed. It's tedious; doing it is a lot easier than saying it. But it fulfills the basic requirements of a program: it is a finite set of stepwise instructions you can follow to create the spreadsheet. You could even read it to someone over the phone.

You might not think of this as a "program" since it requires a human to execute each step; the sequence is not processed automatically by the computer. However, now having defined the spreadsheet solution in English, we're only a step away from converting it into an "AppleWorks program".

Many "programs" these days actually refer to a capability for the use of macros or scripts. We'd normally think of macros as recorded sequences of keyboard actions. To our notion, scripts are series of commands that are entered into a program from a text file, but the distinction between macros and scripts sometimes gets hazy. For example, *TimeOut UltraMacros* allows you to record a series of operations within AppleWorks as you type (a macro), or to define similar actions by entering command words into a word processing document and then "compiling" those commands into a form *UltraMacros* can work from.

Using the *UltraMacros* program, we can take our "spreadsheet generation" instructions above and look in the *UltraMacros* manual for its syntax to duplicate them. As with Applesoft, part of the operation is adding any other structural components the program requires.

The *UltraMacros* compiler requires the word "start" at the beginning of the macro definitions. Following that, there is the macro key to be defined: we used "<ba-S>" ("both apple ess") to indicate the macro will be triggered by pressing the open-apple and closed-apple keys and the "S" key simultaneously. Following that is a colon, then the AppleWorks module that the macro will work within (we used "<all>" meaning our macro can be called up from any module). The rest is all instructions the macro is to carry out, ended with a "!" to mark the end of the macro for the compiler.

UltraMacros has other syntax requirements. Certain command words used by UltraMacros must appear within angle brackets ("<" and ">"). There are special command abbreviations like "oa-Q" (open-apple-Q) to indicate the typing of an AppleWorks command key (open-apple-Q brings up the "quick switch" menu that allows changing to a different file on the desktop). Some keys that aren't represented by printable characters have special names (like "rtn" for the Return key, or "right" for the forward arrow). Where there are several command words (also called "tokens") in succession, they can be placed within a single pair of angle brackets with a colon placed between commands; that is, "<all><oa-Q><esc>" and "<all:oa-Q:esc>" would be treated the same.

"White space" (non-printing characters used to format the file, including spaces, tab characters, and return characters) within brackets is ignored. Characters within curly brackets ("{" and "}") within "<" and ">" are also ignored; this allows us to enter "comments" to explain the macro without the comments interfering with interpretation of the macro. Return characters at the end of lines are also ignored (you need to use the "<rtn>" token to generate a return character as if it were typed from the macro).

Characters typed outside of the brackets are sent to AppleWorks as if they were typed at the keyboard. That is, "1 <rtn>" would be treated as if you typed the 1 key, the space bar, and the Return key in succession.

```
start
<ba-S>:<all         {use from any module}>
<oa-Q               {put up quick-switch}>
<esc                {escape to main menu}>
1<rtn               {"Add"}>
5<rtn               {new file for SS}>
1<rtn               {from scratch}>
Sample.SS<rtn       {file name}>
First<down          {text for cells}>
Second<down>
Third<down>
<right              {next column}>
"========<          {label}>
<down>
@AVG(B1...B3)<rtn>
!                   {end of macro}
```

Okay, it's a little intense trying to boil down a programming lesson into a few paragraphs. But the point is that if you can understand the English version above, with a little work and our few hints above you should be able to understand the *UltraMacros* "program". (When we surveyed users last year, *UltraMacros* was the leading "programming language" in use by our customers.)

Other macro and scripting environments (from Roger Wagner Publishing's *MacroMate* to the script languages of communications programs such as *Talk is Cheap*, *Proterm*, and *Point to Point*) have different syntax, and the responsibility of learning the syntax falls upon the user wanting to customize the program. If you start with a clear idea of the problem in your own words, you should be able to work out a translation into the target language by expending some effort. And as you learn the syntax, the chore will become easier and more automatic, to the point that experienced programmers often sit down and start writing in the target language.

Hypermedia: the next frontier? Even discounting the addition of sound and video to hypermedia programs, just the incorporation of the non-linear linkages and graphical elements usually employed in a hypermedia "program" makes it more complicated to describe in purely textual terms. Since we have to think in textual terms first to write our native-language description of what we'd like to incorporate into a program, descriptions of hypermedia programs can become very complicated even if the individual elements are simple to describe.

For a hypermedia program without explicit scripting capabilities (*Tutor-Tech* and *HyperScreen* on the Apple II, and *Nexus* on the IIgs) the data file itself can still be seen as the program. Program design consists of determining how the various elements of cards, fields, and buttons (files and links for *Nexus*) will combine to control the ways the user can navigate and experience the data.

Both *HyperStudio* and *HyperCard IIgs* have scripting capabilities. But *HyperCard's* use of scripts is integral to the operation of the program while *HyperStudio's* mostly allows control over external commands. The things that this scripting allows you to do make the flavor of the programs different, though there is some overlap.

HyperStudio's out-of-box functionality is so completely encompassed by its visual interface and environment that it is not necessary to use scripting to operate it. With the addition of the Master XCMD from the *HyperStudio XCMD Library Disk, Volume 1,* operations involving XCMDs can be scripted. But since much of *HyperStudio's* internal operation (such as painting, adding buttons, and so on) does not require scripting, it's harder to describe *HyperStudio's* stacks only in terms of "scripts", at least in our "programming" sense.

HyperCard's scripting language, on the other hand, can control all aspects of the program and gives us a way to completely describe the implementation of a stack textually. For example, going back to our "averaging" problem, if we create three fields for input and one for the answer, the code for displaying the average can be placed in the script of a "result" field:

```
on mouseEnter
  put false into nullField
  repeat with i = 4 to 6
    if card field i is empty then put true into nullField
  end repeat
  if not nullField then put average(card field 4, card field 5, card field 6)¬
  into card field 7
  else put "NA" into card field 7
end mouseEnter
```

If we enter valid data into fields 4-6 and move the mouse into the result field 7 (triggering the "mouseEnter" handler above), the average of fields 4-6 will be placed in field 7. (If any of fields 4-6 contains something other than a number, an error dialog will be displayed; we haven't implemented error trapping.)

If we'd like to specify **all** stages in creating the fields on a blank card in a new stack, our "averaging" program can be expressed completely in HyperTalk. In general terms, we save the current tool and pattern, then paint the background. We draw and position seven card fields: three to hold labels, three more to accept our data, and one to hold the result. Then we create a variable holding the same script we used above, add it to the seventh field, and finally restore our original tool and paint settings. Even without a line-by-line analysis of the program it should make sense:

```
on averageCard
  put the tool into ourTool
  put the pattern into ourPattern
  choose bucket tool
  set the pattern to 4
  click at 160,100
  choose field tool
  doMenu "New Field"
  set the rect of card field 1 to 50,50,150,62
  put "First" into card field 1
  doMenu "New Field"
  set the rect of card field 2 to 50,70,150,82
  put "Second" into card field 2
  doMenu "New Field"
  set the rect of card field 3 to 50,90,150,102
  put "Third" into card field 3
  repeat with i = 1 to 3
    set the style of card field i to opaque
    set the lockText of card field i to true
    set the showLines of card field i to false
  end repeat
  doMenu "New Field"
  set the rect of card field 4 to 160,50,310,62
  doMenu "New Field"
  set the rect of card field 5 to 160,70,310,82
  doMenu "New Field"
  set the rect of card field 6 to 160,90,310,102
  doMenu "New Field"
  set the rect of card field 7 to 160,110,310,122
  set the lockText of card field 7 to true
  put "NA" into card field 7
  repeat with i = 4 to 7
    set the style of card field i to opaque
    set the showLines of card field i to false
  end repeat
  put "on mouseEnter" & return &¬
  "put false into nullField" & return &¬
  "repeat with i = 4 to 6" & return &¬
  "if card field i is empty then put true into nullField" & return &¬
  "end repeat" & return &¬
  "if not nullField then put average(card field 4," &&¬
  "card field 5, card field 6) into card field 7" & return &¬
  "else put" && quote & "NA" & quote && "into card field 7"¬
  & return &¬
  "end mouseEnter" & return into newScript
  set the script of card field i to newScript
  choose ourTool
  set pattern to ourPattern
end averageCard
```

We added this handler to the stack script, brought up the message box and typed "averageCard". All that was left to do was to sit back as HyperCard paints the card, creates and positions the fields, sets their properties, and places our script into card field 7. Notice that the syntax here is noticeably like a natural language; eventually, maybe you'll be able to describe what you want to the computer in everyday language and have it created. (Or maybe our "everyday language" will adopt computer jargon by that time.)

The core of hypermedia design will probably redefine how we think of programmers in the future. The 1970's microcomputer programmer dealt largely in paper and cassette tape, machine language, and text-only terminals. In the 1980's, use of graphics became wider and even expected, culminating in the more extensive incorporation of graphics into the user interface on most surviving microcomputers.

In the 1990's, a programmer has to be not only a linguist and rational thinker, but also an artist, poet, motion picture director, screenwriter, and so on. We learned this as our first issue of ***Script-Central*** was assembled; we've added individuals with art experience (Steve "Bo" Monroe) and knowledge of sound (Bruce "HangTime" Caplin) to our stressed-out editorial staff. We still think the "English first, program second" method of development is the right way to go, but the form of the English description is going to start reading less like a checklist and more like a motion picture screenplay, complete with visual and musical direction. And computer programs are going to start looking more like interactive videotapes than blank paper forms reproduced on a computer screen. At some point in the future, a person programming a computer may resemble a director for a stage or motion picture production.

But at every stage, what should be apparent is that program design is not really a function of the computer. It is the definition of the problem in specific instructions that someone else can follow. The implementation of the instructions is where the knowledge of the computer is needed; you may have to revise some of your original instructions to fit them into actions the computer can perform (and the computer language can express), but there's no reason the descriptions of the process can't remain in English.

There is no shame in having no desire to learn how computers work. But if you want to understand books you have to learn to read, and if you want to drive a car you have to get a license. If you want control over your computer system you'll have to learn how to apply logic in terms it can understand or hire a consultant to apply it for you. But please, don't ask for *VCR Plus* solutions; the idea is to release the power of computing to everyone, not to lock it in a box that requires a call to a 1-900-number every time you want to do something interesting.—DJD

Ask (or tell) Uncle DOS

RGB switch

I have an Apple IIe with Applied Engineering's *RamWorks III* and the *ColorLink* option. I also have a Magnavox RGB monitor. I have had a problem similar to Mr. Mastel's ("Apple RGB connections", p. 7.21, April 1991) with programs showing up in grey scale. For me it has been ones from Brøderbund such as *The New Print Shop, Dazzle Draw,* and *The Playroom.* These programs also created so much screen flicker under RGB that they could not be used. Fortunately I can also connect the composite ports and the RGB ones at the same time and switch back and forth between the two as needed via the front panel on the Magnavox monitor. Even my three-year-old has mastered the technique.

Gary Sonnenburg
Omaha, Neb.

Sider/*RamFast* support

C.V. Technologies announced the release of ROM 2 for the *RamFast* SCSI card which supports tape drives. Being an owner of a *Sider C96* and *RamFast,* I was anxious to purchase ROM 2. About a year ago, I elected to give up my tape backup capabilities for the *RamFast*'s unbelievable speed. At that time C.V. Technologies was planning tape drive support with ROM 2. Usually anything that is worth a darn takes a fair amount of time to perfect. In this case it was no exception. When I received ROM 2 it consistently locked up the tape drive...ack!

C.V. Technologies and I probably sent more missiles back and forth than the U.S. and Iraq. Finally, I listened to Andrew Vogan and sent him my *C96.* Andrew spent the better part of two days tracking down the problem. It turns out that the microcontroller on the tape drive was not fast enough to handle the *RamFast*'s warp speed. He called the manufacturer and found that they had a new model that would work. Andrew returned the drive and told me to contact Larry Beyer of B & D Computer Repair in Chicago.

Larry installed a new microcontroller and thoroughly tested my *C96.* The tape drive is now incredibly fast thanks to C. V. Technologies and B & D Computer repair.

If you want great products and service for the Apple II make sure that you use these companies!

G. Scott May
Buffalo Grove, Ill.

B & D Computer Repair's address is 6115 S. Massaholt, Chicago, Ill. 60638, 312-735-9010. Larry has been handling ***Sider*** *repairs for several years.—DJD*

Beagle Compiler and "&"

There's a way for assembly language programs to tell whether they've been called from pure Applesoft or from the *Beagle Compiler,* because I've done it. Just use two "&" characters to start each of your ampersand calls, as is suggested on page 20 of the *Beagle Compiler* manual. Regular Applesoft will gobble only the first "&" character as it joins your assembly language routine. Compiled "Beaglesoft" will gobble both of them. The first thing you do in your routine is JSR CHRGOT (JSR $00B7) to get the current character pointed at by TXTPTR ($B7-B8). If it's an "&", then you're running under Applesoft. If it's not, you're running under the compiler.

Actually, you need to check CURLIN+1 ($76) first to see if you are working in the immediate mode. If it is $FF, then the entry in your "&" code is occurring during immediate mode and not during program execution; you would want to use routines compatible with Applesoft under that circumstance.

You should make extensive use of Applesoft's built-in syntax checking to gather your variables and send back your results, and all will be well.

Craig Peterson
Santa Monica, Calif.

This technique doesn't seem to work with the more recent editions of the ***Beagle Compiler****, but Craig's basic advice is sound: check to see if the compiler or Applesoft is interpreting the program code, then jump to the appropriate "&" handler.*

We found that the CHRGOT routine is different with the compiled code running; you can verify this by checking location $B7. The contents under Applesoft are $AD (LDA $xxxx), under the compiler the location contains "$AC" (LDY $xxxx). Checking that location will if the compiler is running ***with the current version****; this method is neither sanctioned nor guaranteed by Beagle, but the manual gives no "ID bytes" for determining if the compiler is executing the program. (Beagle is looking into a "guaranteed" way for us.)*

The next trick is catching the "&". Applesoft actually converts its keywords into one-byte representations ("tokens") to make the programs more compact. Token values are always in the range $80 to $FF (text embedded in the Applesoft program is represented in the ASCII range $00 to $7F), and the token for the "&" command is $AF. So you need to check for that value rather than the ASCII value for "&" ($26).

When we put the above changes in with Craig's design, our dummy subroutine to test parsing a double ampersand looks like this:

```
CURLINE  GEQU  $0075         current line number
CHRGOT   GEQU  $00B7         get current character
TXTPTR   GEQU  $00B8
COUT     GEQU  $FDED

         ORG   $0300

Entry    START
         lda   CURLINE+1
         cmp   #$FF          immediate?
         beq   Immed
         jsr   CHRGOT
         cmp   #$AF          AS token for "&"
         beq   Interp
         bne   Compiler

Immed    ldx   #0
Immed1   lda   msg1,x
         beq   Interp
         jsr   COUT
         inx
         bne   Immed1
msg1     DC    H'8D',C'Immediate mode...  ',H'8D00'
Interp   ldx   #0
         jsr   BumpPtr
         lda   (TXTPTR,x)
         ora   #$80
         jsr   COUT
         lda   #' '
         jsr   COUT
Interp1  lda   msg2,x
         beq   Exit
         jsr   COUT
         inx
         bne   Interp1
msg2     DC    H'8D',C'Applesoft interpreter',H'8D00'
Compiler lda   #$87          beep twice
         jsr   COUT
         jsr   COUT
         ldx   #0
         lda   (TXTPTR,x)
         ora   #$80
         jsr   COUT
         lda   #' '
         jsr   COUT
Compilr1 lda   msg3,x
         beq   Exit
         jsr   COUT
         inx
         bne   Compilr1
msg3     DC    H'8D',C'Beagle Compiler',H'8D00'
Exit     jsr   BumpPtr       move past "X"
         clc                 make A/S happy
         rts                 return

BumpPtr  inc   TXTPTR
         bne   PtrOK
         inc   TXTPTR+1
PtrOK    RTS
         END
```

Line 30 in the following BASIC program installs and calls the routine (the X in line 30 is not used as a command, just echoed by the subroutine):

```
10 PRINT CHR$ (4);"BLOAD AMPER": REM at $0300
20 POKE 1013,76: POKE 1014,0: POKE 1015,3
30 & & X
```

There are still problems the programmer will need to resolve. First, safe memory has to be found for the routine (safe with Applesoft or the ***Beagle Compiler*** *environment). Second, support routines are different in the two environments; Applesoft's routines are in ROM, of course, and the* ***Beagle Compiler****'s routines are part of the COMPILER.SYSTEM file loaded to run compiled programs. If your ampersand routines need to call these support routines, the addresses for Applesoft and the* ***Beagle Compiler*** *will be different.*

The support routine addresses and functions for the ***Beagle Compiler*** *are located in the compiler's manual. Apple's Applesoft documentation doesn't list Applesoft internal routines, but one source is* ***Assembly Language for the Applesoft Programmer*** *by C. W. Finley, Jr., and Roy E. Myers. (Addison Wesley, ISBN #0-201-05209-1).—DJD*

Boo, hiss

My IIgs system includes a ROM 01 CPU, a *TransWarp GS* with 32K cache, an Apple High-Speed SCSI interface, and an Applied Engineering *GS-Ram Plus* with 3 megabytes installed.

I recently received version 0.95 of *SoundSmith* from Huibert Aalbers. Since my onboard speaker doesn't do justice to the music this software produces, I plugged a portable stereo speaker into the computer's headphone jack. This works fine when the music is playing, but there is a very audible hiss when the music is off. I realize the IIgs's internal speaker makes the same hiss, but it is pretty much drowned out by my *System Saver*'s fan.

Is there a practical way to reduce or eliminate the noise the computer produces? In the September 1989 newsletter ("Apple upgrades IIgs hardware") you wrote that the ROM 03 IIgs's have "cleaner sound". Does this mean newer IIgs's have less hiss?

Is there a better way to connect a single speaker to the IIgs than using the headphone jack? I don't want to invest in a sound card and stereo speakers, but a single amplified speaker is worthy of consideration.

Stephen Gant
Manteca, Calif.

Part of the hiss problem on ROM 01 machines has to do with the circuitry layout on the motherboard. Any circuit carrying electronic signals tends to act as a broadcast antenna for those signals; if another circuit lies within the "broadcast region" some of the signal can "cross over" to the second circuit (this type of interference is referred to as "crosstalk"). The audio circuits on the ROM 01 machines are close enough to certain other circuits to pick up interference from them. This can't be changed without altering the motherboard; it was alleviated on the ROM 03 motherboard by changing the circuit layout to better isolate the audio circuits from possible interference sources.

I have a ROM 01 at home and haven't found the hiss to be that objectionable. The easiest way to reduce it is to lower the treble response on your speaker system; I've found this doesn't seem to affect the sound quality too extensively.

*A stereo card may help; I do notice less hiss on my **Sonic Blaster**'s output.—DJD*

More GS/OS versions

In the latest *APDALog* (April 1991), there is no mention of Apple IIgs System 5.0.3. Is this an unreleased version which is basically the same as System 5.0.4? This would also mean that 5.0.3 only works for a IIgs with one megabyte of memory and above, I presume.

A. Hadiwidjaja
Ashfield, N.S.W.

System 5.0.3 was only "semi-released"; it never made it to Apple dealers, for example. A few obscure problems in 5.0.3 forced a revision to version 5.0.4, which was released a few months later.

A convenient way to break down the IIgs System Software versions is by system requirements. System 3.2 was the last version that would run on a basic 256K machine, but its use can't be encouraged since the IIgs toolsets have been revised drastically since it was introduced.

System 5.0.2 will operate on most 512K machines; the use of AppleShare or the addition of the newer SCSI drivers provided with the High-Speed SCSI Card support disk may push the memory requirements slightly over that.

For systems with one megabyte of memory or more, System 5.0.4 is the "current" recommended system software. IIgs users who have less than one megabyte of memory should strongly consider upgrading; memory prices are very reasonable now, and use of the current system software version is necessary to have access to software that uses features found only in the newer versions. We generally recommend IIgs owners aim for at least two megabytes of memory to give themselves some "breathing room".—DJD

900 number gotcha

After reading the letter from Mike Sample in the June 1991 issue, I would like to point out a problem with 1-900 lines that many people do not know exists. Due to a change in Texas law to prevent abuse of 900 and 976 numbers, telephone companies that cannot block service to these numbers on a customer by customer basis must completely block service to these numbers. Being served by such a telephone company, I am therefore legally blocked from calling Applied Engineering's or any other 1-900 number. This affects many users who live in rural areas where there is already no convenient dealer or user group.

As a professional systems analyst/programmer who has used Apple computers since 1980, I fortunately have had little need to use telephone support lines. However, I would encourage companies to rethink their use of 1-900 numbers so that others served by telephone companies affected by this law will not lose one of the few resources available to them.

Charles W. Hall
Comfort, Texas

*The use of 900 numbers is generating controversy for all types of computer products. William F. Zachmann's editorial in the July 1991 (Vol. 10, No. 13) issue of **PC Magazine** is "For a Good Time, Call 1-900-SUPPORT". He looks at the issue critically, but misses your predicament.*

One of the other bizarre developments is a mail-order house that uses a 900 number for product orders. The company claims that this allows them to sell products at a lower price by offsetting sales support costs. Those who ask the most questions pay the most in telephone charges.—DJD

Disk fading

I'm thinking about taking the plunge and buying a hard drive. I went back through a couple of years of ***A2-Central*** and couldn't find the answer to this question.

All of the information on the disk is magnetically imprinted, including the low-level formatting. Some of this information will not be called up and renewed for long periods of time, maybe years. Will this information gradually fade and cause disk errors? If this is a problem, could a layman save everything he had on the disk including the high-level formatting, renew all the original formatting and put Humpty Dumpty back together again? Maybe there is no problem with magnetic fading?

I could stand to renew the formatting every other year better than introducing, say, a 0.005 per cent error factor. With the millions of operations that go on in the computer every boot and run, I have visions of my software slowly developing bugs and glitches after a few years. I have had a few programs on floppies go bad slowly. You can recover from one floppy; I shudder to think about the same thing happening to a hard drive.

Rollin Ratchen
Salem, Oregon

We'd like to reduce your paranoia some. Just some, not entirely; you can't see what's happening to your data, so it's better to be very safe rather than very sorry.

Technicians have told us that computer data on disk media may be good for ten years or more, so disks should be re-written (all data backed up, the disk re-formatted, and the backup restored) at least that often.

We were also told magnetic tape should be rewritten at least every five years. Since magnetic tape is stored as "layers" wound on a spool, tape has the additional problem of "print-though", where the magnetic image on one layer can try to imprint itself on the adjacent layer, disturbing the original information. (This happens on audio and video tapes, too, but audio and video tape recorders don't come to a crashing halt if their data is slightly distorted.)

There are other things in our environment that cause magnetic fields that can corrupt data on magnetic media. Hard disks are resistant to most minor disturbances (otherwise you'd have trouble using them close to your computer and monitor), but if you have the hard disk setting on the floor and start an electric motor next to it (say, a vacuum cleaner) the magnetic field exerted by the motor may trash your data.

Also, you have to remember that the hard disk itself is a mechanical device with moving parts that may wear out long before the data deteriorates. A hard disk with a "MTBF" (mean time between failures, or the average time that can be expected to pass before the drive fails) of 50,000 hours should have an average lifetime of between five and six years. If you wait ten years to "reconstitute" it, you're taking chances.

Finally, you have to assume that nothing else damages the drive. It's always possible that a power surge, rampaging program, or physical damage could take out the drive at any time.

*So the rule is simple: **back up your data regularly**. In case of a failure at any time, you won't lose any more work than was completed since your last backup.—DJD*

A flash of *SuperView*

I found a bug in *SuperView*, the graphic viewer you put on your May 1991 disk. With AppleTalk enabled, the pictures are interrupted with flashes. Disable AppleTalk and everything works fine again.

Jack van Soest
Vlaardingen, Netherlands

***SuperView** is a utility that allows viewing IIgs pictures using up to 3200 colors. The IIgs is normally limited to displaying 16 palettes of 16 colors (up to 256 colors) when the screen is set to use the 320 by 200 resolution mode.*

In order to get 3200 colors, the IIgs palettes are altered "on the fly" as the various lines are displayed so that a different 16 colors can be displayed for each of the 200 scan lines (16 times 200 is 3200 colors). As you can imagine, continuously altering the palettes as the picture is displayed takes a lot of work; the processor is kept pretty continuously busy.

AppleTalk also insists on having access to the processor occasionally so that it can check on the network status. The "flash" is caused by the handling of an AppleTalk interrupt causing the 3200-color display code to get "out of sync" in updating the palettes for the displayed image. This could be prevented by disabling interrupts, but this could cause the network to assume your IIgs had been disconnected and quit looking for it; not a good thing, especially if you're using an AppleShare volume as a primary disk drive.

*So it really isn't a "bug", it was a design decision. Chris McKinsey (**SuperView**'s author) decided that leaving you connected to the network was more important than giving you flicker-free pictures. This only affects AppleTalk users, and is a limitation of the resources of the IIgs.—DJD*

MD-BASIC and C

For those few (like myself) who have the need to convert 16-bit C programs to run on 8-bit Apples—try *MD-BASIC*! The latest version supports whiles, repeats, if-then-else, etc., and it is fairly easy to use macros to convert printf("\nHello") to PRINT "Hello". I now have a fairly large program running under GS/OS and nearly the same program running under Applesoft!

Richard Phares
Annapolis, Md.

Apple's Mac LC "solution"

I just received the July issue of ***A2-Central*** and would like to comment on a couple of points. On page 7.44 you say, "...But given the limitations of the LC as a host, the erratic commmittment of the parent company to support its installed base of users..." Great paragraph!

In my opinion the Mac LC (and the Classic) are the biggest pieces of "junk" ever placed on the market. The LC, I believe, will be Apple's "Edsel." My church bought one over my severe objections and are already sorry for the purchase and looking for a way out. It's terrible to have to tell your minister that "I told you so."

On page 7.45 in Ask Uncle DOS, Robert Halls letter title "Revenge" hit the nail somewhat on the head. I don't agree completely with some of his comments nor your response. I am no longer an Apple'er after 10 years of Apple IIe's, IIc's and IIgs's. I have sold almost all of the hardware and a large portion of my extensive software and reference libraries. (Thank goodness there is still a fairly good resale market for this "stuff".) I am working from an IBM compatible 386SX clone which sports 4 meg of memory, a math coprocessor, a 3.5 drive, a 5.25 drive, an 80 meg internal hard drive, a 2400 baud internal modem and a Hewlett Packard *DeskJet 500*. By the way this system cost me a "heck of a lot less" than a comparably equipped Macintosh LC would have. I run with MS DOS 5.0 and Windows 3.0. I have been running this system for over a month using *WordPerfect 5.1* (just waiting for Windows version), *Excel 3* for Windows (WOW!), *dBase IV* and *PageMaker 4.0* for Windows. Although I have suffered some "discomfort" with the learning curve, I must admit that at this point I DO NOT MISS APPLE, including AppleWorks which I did not think I could ever do without.

Revenge—I have had mine. I am no longer an Apple customer nor will I ever be again. Not only that I am sharing my conversion experience with many other former-to-be Apple II owners. I really think it is too late for Apple to do anything about the alienation of the significantly large Apple II customer base. I think the 40% per cent drop in their stock value should say something.

I know that Apple is ***A2-Central***'s bread and butter and this type of letter does not bode well for the future, but I dare you to print it. I will be watching ***A2-Central*** until my subscription expires next May. I hope to see it in print...

Raymond W. Crowley
Manchester, Mo.

Well, I'm not too fearful of printing it since obviously we have a great many Apple II people who don't feel the same way. I too have my MS-DOS systems, starting with a laptop to do what Apple couldn't seem to achieve; I still think the Apple II's problem is unrealized potential rather than lack of potential.

Apple's stock has been playing ping-pong for the last three years, usually boosting after new product introductions and then slowly fading toward the Christmas buying season when the Apple II (which has been poorly marketed of late) was formerly the traditional seller. The intro of the new Macs last fall sent the stock zooming and now it has moderated; let's hope the new Consumer Products group will have something to say about the Apple II's role this season.—DJD

Good reasons

I enjoy reading your newsletter, *even though I own a IIe and find the number of germane articles diminishing.*

Give me three reasons I can present to my wife for upgrading to a IIgs. If she buys at least two of them, I will be both grateful and less inclined to fire off letters with snide italic passages.

Robert W. Hughes
Reynoldsburg, Ohio

We've covered several of the issues in the past, with software being a weak link, so let me give you three software reasons to upgrade: hypermedia, programming languages, and networking. Your decision to actually change systems will probably depend on your interest in these areas.

*The IIgs has the basic hardware to support powerful hypermedia programs like **Nexus**, **HyperStudio**, and **HyperCard IIgs**, giving it perhaps the most potential in this area. The IIgs's individual capacities for sound, color graphics, megabytes of memory, and so on may not be the most intimidating on the market, but it is an affordable way to get the combination of these capabilities in a ready-to-use package. If you're curious about the IIgs, these programs are worth investigating (we've spoken about them often ourselves).*

The IIgs provides a richer environment for classic programming languages such as Pascal and C, where Apple II implementations have stuttered. Your choice of languages for the IIgs is broader, and 8-bit languages are also available, of course. In addition, if you like the idea of writing using a graphical user interface, Apple has already made many of the routines available as part of the IIgs system software's tools.

Finally, although Apple also brought networking capability to the IIe via the Apple II WorkStation Card, the IIgs handles networking more naturally. The IIgs Finder supports network volumes automatically (as should all good IIgs programs) and has the extra resources to allow the use of network printer drivers as part of the operating system. And, although not strictly networking, the IIgs also has more flexible support for non-ProDOS file systems through GS/OS, already evidenced by the support of AppleShare and High Sierra file systems.

*It isn't difficult to point to other computers and find chinks in the IIgs armor, but taking **all** things into account at once the IIgs fares better. HyperCard on the Mac isn't in color, MS-DOS systems have to kludge the use of extended memory . Neither system has the built-in sound capability of the IIgs.*

We don't know whether any of the above three areas interests you but it's our belief that hypermedia and networking capabilities are becoming desirable features of computers, and that although not everyone wants to be a system programmer, the availability of better languages will attract the better programmers (most programmers who write GS/OS-based software are not anxious to return to the 8-bit programming environment).—DJD

A2-Central™

© Copyright 1991 by
Resource-Central, Inc.

Most rights reserved. All *programs* published in ***A2-Central*** are public domain and may be copied and distributed without charge. Apple user groups and significant others may obtain permission to reprint *articles* from time to time by specific written request.

Publisher: **Tom Weishaar**
Editor: **Dennis Doms**

with help from:
Sally Dwyer, **Dean Esmay**, **Joyce Hammond**, **Jay Jennings**, **Jeff Neuer**, **Ellen Rosenberg**, **Denise Shaffer**, **Tom Vanderpool**, **Jean Weishaar**

A2-Central,—titled ***Open-Apple*** through January, 1989—has been published monthly since January 1985. World-wide prices (in U.S. dollars; airmail delivery included at no additional charge): $28 for 1 year; $54 for 2 years; $78 for 3 years. All back issues are currently available for $2 each; bound, indexed editions of our first six volumes are $14.95 each. Volumes end with the January issue; an index for the prior volume is included with the February issue.

The full text of each issue of ***A2-Central*** is available on 3.5 disks, along with a selection of the best new public domain and shareware files and programs, for $84 a year (newsletter and disk combined). Single disks are $10.

Please send all correspondence to:

A2-Central
P.O. Box 11250
Overland Park, Kansas 66207 U.S.A.

A2-Central is sold in an unprotected format for your convenience. You are encouraged to make back-up archival copies or easy-to-read enlarged copies for your own use without charge. You may also copy ***A2-Central*** for distribution to others. The distribution fee is 15 cents per page per copy distributed.

WARRANTY AND LIMITATION OF LIABILITY. We warrant that most of the information in ***A2-Central*** is useful and correct, although drivel and mistakes are included from time to time, usually unintentionally. Unsatisfied subscribers may cancel their subscription at any time and receive a full refund of their last subscription payment. The unfilled portion of any paid subscription will be refunded even to satisfied subscribers upon request. OUR LIABILITY FOR ERRORS AND OMISSIONS IS LIMITED TO THIS PUBLICATION'S PURCHASE PRICE. In no case shall our company or our contributors be liable for any incidental or consequential damages, nor for ANY damages in excess of the fees paid by a subscriber.

ISSN 0885-4017
GEnie mail: A2-CENTRAL
Voice: 913-469-6502
Printed in the U.S.A.
Fax: 913-469-6507