

Launching into BASIC

Joe Abernathy

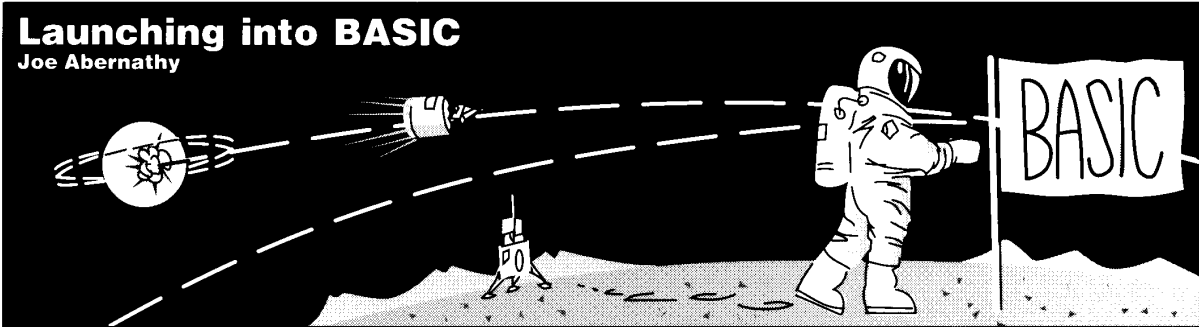


Illustration by Robert Williamson

BASIC is back for the Apple IIGS. Dressed in its Sunday best, polished with modern capabilities, BASIC is better than ever, to the extent that you can be within an hour of having your first Apple IIGS desktop-based program up and running.

Welcome to "Launching Into BASIC," *Call -A.P.P.L.E.'s* new column on BASIC programming for the Apple IIGS. We will explain and demonstrate everything you need to create sophisticated desktop applications, and along the way we'll build a significant source code library that will put you ahead of the pack.

It has been easy to forget that the IIGS even has a built-in BASIC prompt, since you have turned to assembly, Pascal or C to use the Toolbox. (See Toolbox sidebar.) But there now are two strong commercial BASICs, with one or two more in the works.

Apple came first with GSBASIC, but recently put it out of its misery. TML BASIC from TML Systems is actually the product Apple likely had in mind. (If you have any GSBASIC source code on hand that you want to save, it can easily be ported to TML BASIC.) Absoft Corp. offers AC/BASIC, whose programs are compatible with Microsoft BASICs. Micol Advanced BASIC for the IIGS was recently introduced, but we have not had the opportunity to examine it. During the first quarter of 1989, Byte Works reportedly will publish ORCA/BASIC.

If you have not yet purchased a BASIC compiler, your choice is pretty straightforward. Choose AC/BASIC if you have little or no knowledge of IIGS programming, for it is the friendliest to learn. Choose TML BASIC if you already have some idea of what the Toolbox is about or don't mind learning. The difference is one of assumptions in the compilers' design—TML chose to use very standard (to APW programmers) hooks into the Toolbox, while Absoft offers Toolbox access in the form of built-in statements such as WINDOW (at the price of reduced flexibility).

Tools of the Trade

Central to everything is the Toolbox, the set of some 900 routines built into the IIGS that together make Apple's graphical interface available to programmers. The Toolbox is what sets

Toolbox

Keep in mind the BASICs for the IIGS use the ProDOS 16 world on the IIGS. This means full toolbox capability is available through coding that may only involve a single line of code up to many lines of code in an assembly language module. If you want to do some toolbox work using Applesoft and ProDOS 8, perform some serious reading and research. Not all tools can be accessed while under the umbrella of ProDOS 8.

Cecil

the IIGS apart, but it also is what has made IIGS software so slow in arriving — the style of programming needed to employ tool calls is essentially foreign to programmers who learned on the old Apple II line.

This, however, like prejudice of any kind, is exposed in the end to be wrong-headed. You will come to regard the Toolbox as a good friend once you have made its acquaintance. Desktop programs, which is what programs written using the tools are generally called, mainly consist of very similar routines that can readily be reused. The result is programs that are cleaner, more sophisticated, and both easier and faster to write.

How is desktop programming different from the way you've always programmed? Aside from an aura of mystery, not much. The only thing new you really have to learn is how to think in terms of an event-driven program, instead of the "top-down" style encouraged by old Applesoft BASIC. (See Top-down sidebar.)

You also will notice that line numbers and GOTOs are in short supply these days, their place taken by labels and control structures such as IF-THEN-ELSE and WHILE-DO. If you've worked in C, Pascal, or another structured language, you're a step ahead of the game; if not, you're in for a pleasant surprise.

Cecil recently lamented the demise of GOTO in the Consultant's Corner, and his feelings are understandable. But its passing is actually a cause for happiness. GOTO was a spoiler constantly at work making source code difficult to read and impossible to debug. Control structures make it quite clear what turns the program is allowed to take and when. They encourage you to carefully think through a routine, therefore, things often turn out right the first time.

We will tie these common themes together, and give you a standard programming platform, by designing two program shells, one for each language. What these shells do is handle all of the background work that has to be done to prepare the tools for use, and set up the desktop environment and an event-handling routine. The shell can be the foundation upon which you build all of your subsequent programs — all you do is add the specific routines needed to accomplish the task at hand. Subsequent source code presented here will be designed to use within these shells.

AC/Exerciser is the shell for AC/BASIC. In addition to setting up the desktop, it shows how to implement the most common file-handling techniques, and some features specific to AC/BASIC. Space limitations require that the TML shell await the next column.

Getting Down to Basics

The true measure of any language software is found when you sit down to do some programming. But since there are two sophisticated, incompatible dialects of BASIC available for the Apple IIGS, it is possible to examine the programming issues of only one at a time.

Top-down

Top-down programming is a method or procedure whereby one starts writing a program and develops subroutines as needed. The more formal definition is a method or procedure that starts at the highest level of abstraction and proceeds toward the lowest level.

Contrast this with structured programming. In a loose sense, this involves subroutines. In simple terms, one starts with the simplest code or subroutine, works with code or subroutines that call the simplest subroutine, etc. In a sense, this is bottom-up programming starting at the lowest level of abstraction and proceeding toward the highest level.

Cecil

What follows is an introduction to the capabilities and limitations of AC/BASIC, and of the program AC/Exerciser. We will discuss the specific routines within AC/Exerciser in a moment.

AC/BASIC is potent. To give you an idea of just how potent, it took less than an hour from first booting the compiler to get an old Applesoft BASIC program running as a windowed desktop application. Two evenings later, it was a fully modern program featuring buttons and Super Hi-Res (SHR) graphics.

AC/BASIC is published by Absoft Corp., the company behind Microsoft's BASIC for the Macintosh as well as AC/BASIC for the Amiga. Hence, the source code of these three products is very machine-portable. For Apple II enthusiasts, this raises the intriguing prospect of doing cross-development for the Macintosh on a IIgs.

Machine snobbery aside, portability truly is a strong point, but only one of many. If you like BASIC, you will love AC/BASIC. It is fast to learn, quick in operation, and creates true stand-alone GS/OS programs. It jettisons the traditional mindset that says you must start with a text-based user interface and work your way up to the desktop: AC/BASIC programs automatically compile as windowed desktop programs. (The Apple II 40- and 80-column text screens are not available.)

Some of the significant features include: SOUND, WINDOW, PALETTE, PICTURE, BUTTON, MENU, IF-THEN-ELSE and WHILE-WEND. Twenty-five QuickDraw II operations are available, such as OVAL, RECT, FILLRECT, ERASERECT, ROUNDRECT, etc. Programs will compile in 320 or 640 mode, and REPEAT will cause music or sound to sequence in the background automatically. This is good for games.

Advanced features include the ability to INCLUDE external source code from your programmer's library; a machine language subroutine interface (which allows direct toolbox calls); and custom wave forms for the Ensoniq sound chip (DOC). Dynamic arrays, IEEE and decimal math, and 32- and 64-bit floating point support are provided.

Absoft gave careful thought when designing the tools you use to maintain the desktop. There are event handlers for the mouse, menu bar, timer, errors and dialogs — everything you might need. ON MENU GOSUB and MENU ON, for example, catch any mouse clicks in pull-down menu items. AC/Exerciser shows how event handlers are used (nowhere else will you find a main event loop two lines long).

Line numbers are not needed and both numeric and text labels are available for identifying subroutines. The programs you write will share the indented flow and readability of structured languages. IF-THEN-ELSE, WHILE-WEND and

FOR-NEXT handle control of block functions.

Programs are typed in via a text editor. Absoft licensed the text editor used in APW, but chose to rewrite it in a fashion that actually impairs its functionality. You cannot rename your source file as you are editing it, and you likewise cannot copy source files to different volumes. In addition to the inconvenience, these failings undo whatever benefits batch compiling might have afforded.

The text is turned into a program by running it through a built-in, standard two-pass compiler. The compiler is fast, and ostensibly supports batch jobs. One problem with the compiler is that it is pretty simple-minded in identifying anything other than an outright syntax error.

Compiling creates stand-alone GS/OS programs that can be launched by the Finder. Actually, you can create programs that rely on AC/BASIC being installed in the system folders of the disk you boot from, or you can compile a program as a true stand-alone application. Compiling for stand-alone, that attaches the Absoft run-time libraries to your program, increases program size by 75K. (See Memory sidebar.) There is no charge for this use of the libraries, although Absoft requires that you give them a plug somewhere in your program.

A significant limitation in the current release of AC/BASIC is that it is difficult to access any Toolbox functions that are not directly implemented in the language. You can do so if you have an understanding of assembly language, but it doesn't follow that BASIC programmers are going to want to fool with assembly. Like the lady said, that old dog won't hunt in Texas.

Exercising the Options

AC/Exerciser is the shell for AC/BASIC mentioned earlier. It is actually a complete program that lets you directly issue some of the basic calls your applications will be likely to use. Options are chosen from a standard menu bar; you can enter the parameters of a call or accept the defaults that are provided.

AC/Exerciser shows how easily you can create a professional-feeling application in this language. Absoft's metaphors are truly elegant — they do the job right without losing any of the accessibility of BASIC.

To get AC/Exerciser running, type in the code in Listing 1 and compile it with the U option clicked (no default window). Create a volume off of your root directory called /FW.PROCS/, then type in the two files AC.EX.ABOUT (See Support File 1) and AC.EX.HELP (Support File 2) and save them to that volume. (Alternately, you can put them on any volume if you edit the subroutines at 120; and 130; to reference the correct volume name.) To save you some typing, AC/Exerciser also is available for electronic download — see the heading entitled Resources.

Only six lines of code are required to implement the program's setup and event loop. The subroutine SETUP establishes values for the screen size (to aid in program portability to machines with different screen dimensions). The SETMENU subroutine customizes the pull-down menu bar.

The ON MENU GOSUB statement alerts the program that a subroutine is available to interpret custom menu events, and MENU ON activates the menu event handler. The MAIN event loop causes the program to simply idle, waiting for menu events to occur. (New Desk Accessories (NDAs) under the Apple menu are handled without your interference. So are the FILE and EDIT menus, unless you choose to customize them.)

The routine MENUPROC, installed by ON MENU GOSUB, reads a menu event and its item number, then routes

control to an action subroutine.

10: and 20: show how to open, read, print and close a file. For print operations, you can "open" the printer for output of a file, and the familiar printer dialog boxes will be generated for you. The command FILES will generate the familiar desktop file selection dialog box.

The routine at 30: shows how to kill a file. You also can rename files, but any more advanced file manipulation becomes difficult. For more sophisticated applications that require internal data files, a variety of routines such as GET and PUT support input/output.

WINDOWS lets you create various sizes and styles of windows in various places on the screen. Be careful not to exceed the suggested defaults or you might create a window off the screen and force yourself to reboot the computer.

The SOUNDER option lets you work with the AC/BASIC SOUND statement. The AC/BASIC manual, generally well-written and usable, fumbles where the sound routines are concerned, making them difficult to use beyond the most basic level. Digitized sound files cannot readily be loaded, but SHR pictures can.

HELP and ABOUT.. show file access and mouse-polling. The routines under the SPECIAL menu allow you to manipulate the font, color, size and style of text.

Notably lacking in AC/Exerciser is a demonstration of the built-in graphics commands. These were left out because most of them require manipulation of data arrays, which is not especially difficult but is often hard for beginners to understand in this or any other BASIC. Good demos of the graphics can be found on the AC/BASIC disk itself.

Call -A.P.P.L.E. Update

AC/BASIC was formally reviewed by Ken Kashmarek in the May 1988 issue. Ken reviewed an early version of the language and listed a couple of concerns he felt should be addressed.

AC/BASIC will only print via the Apple IIGS ImageWriter driver. This is slow through no fault of Absoft, and the print quality is unacceptable by any professional standard. Absoft has now added the ability to print in a faster draft mode, but any program that prints in GS native mode is going to have drawbacks until such time as a more capable driver is provided as part of the IIGS system software.

Another new concern is that printing will not work at all in

conjunction with system software versions 3.2 and 4.0 (GS/OS). Absoft's interim solution is to use the printer driver from your 3.1 system software, which should work without hitches.

Ken also pointed out that AC/BASIC doesn't truly launch programs after a compile & run command, with the result that programs return to the Finder, from which you can relaunch AC/BASIC to make changes in your program. This adds quite a bit of time to development turn around. This issue has not yet been directly addressed, but it is feasible to work out of /RAM5/, particularly if you have APW (write two short APW shell utilities, one to load AC/BASIC to RAM, and another to run it from RAM).

Summary

Today is the day to get started programming your IIGs. The tools you've been waiting for are now in place, and winter grips the land, so you can't beg off for the beach. A tremendous adventure is in store, with the full promise of the Apple IIGs as your reward.

Stay tuned for TML BASIC and how to update GSBASIC code into TML and the A.P.P.L.E. Filer, in our next installment.

Resources

AC/BASIC: \$125, Apple IIGs with 512K and 800K floppy drive. Absoft Corp., 2781 Bond St., Auburn Hills, MI 48057. (313) 853-0050.

TML BASIC: \$125, Apple IIGs with 512K and 800K floppy drive. TML Systems Inc., 8837-B Goodbys Executive Drive, Jacksonville, FL 32217, (904) 636-8592.

Micol BASIC: New product. Micol Systems, 9 Lynch Rd., Toronto, Ontario, Canada MSJ 2V6, (416) 495-6864.

ORCA/BASIC: This product has been pre-announced by The ByteWorks on AppleLink, and will be available during the first quarter of 1989.

The Cortland Project: AC/Exerciser available for download. Electronic posting of questions for the *Call -A.P.P.L.E.* Consultant's Corner, with support for assembly, Pascal, BASIC and C. Europa BBS, free, 1200/2400 bps, (713) 526-0714.

A Review of AC/BASIC, Ken Kashmarek, *Call -A.P.P.L.E.*, May 1988.


The Apple II BASIC Handbook, Douglas Hergert, Sybex, Berkeley, CA.

Basic Apple BASIC, James S. Coan, Hayden Book Co., Hasbrouck Heights, NJ.

Programmer's Introduction to the Apple IIGs, Apple Computer Inc., APDA, Renton, WA.

Apple IIGs Toolbox Reference Volumes 1 and 2, Apple Computer Inc., APDA, Renton, WA.

Mastering the Apple IIGs Toolbox, Dan Gookin and Morgan Davis, Compute! Publications Inc., Greensboro, NC.

Note: The above APDA publications also are available from Addison-Wesley Publishing. 

Memory

No matter how you cut it, when compiled programs are executed, they need more memory than interpretive BASIC such as Applesoft (whose support code is in ROM) or an assembly language program. Compiled code involves a lot of calls to subroutines that come from what is known as run-time libraries.

What Joe is saying is, if you are working with a disk under the control of AC/BASIC, when you ask to have the program executed, the operating system loads in the necessary subroutines if they are not already in memory. In a stand-alone environment, you tell the compiler to create a program file that not only contains the program code, but also, the necessary support code from the run-time libraries.)

Cecil

Listing 1 AC/Exerciser

```

' -----
' AC/Exerciser
' By Joe Abernathy
' Version 1.0, Sept. 19, 1988
' (C)1988-89, Call-A.P.P.L.E.
' All Rights Reserved.
' This program may be distributed and used
' freely, but may not be included
' in any commercial package without the
' written consent of Call-A.P.P.L.E.
' You may use any part or all of this
' program as a foundation shell.
' Portions of this program include
' material copyrighted (C) by Absoft Corp.
' 1988. Used with permission. All
' other copyrights acknowledged.
' -----
' File: AC.EXERCISER
' Compiler: AC/BASIC for the Apple IIGS.
' Portability: Amiga BASIC; Microsoft BASIC
' Compile with the U option selected.
' -----

' Customize the ABOUT box:
' $About "AC/Exerciser, by Joe Abernathy"

gosub setup          ' Setup defaults
gosub setmenu        ' Setup menu bar
on menu gosub menuproc ' Setup menu event
menu on              ' trapping.

main:                ' This program is
goto main            ' event-driven.

' -----
' This is a template for reading and acting
' on pull-down menu events. Think of it as
' the main event loop:

menuproc:            ' Trap menu events
menunum = menu(0)    ' Read which menu
itemnum = menu(1)    ' Read which item
if menunum = 3 then  ' FILE menu chosen
  if itemnum = 1 then ' READ chosen
    gosub 10          ' So gosub READ
  elseif itemnum=2 then ' PRINT chosen
    gosub 20          ' Gosub PRINT
  elseif itemnum=3 then ' DELETE chosen
    gosub 30          ' Gosub DELETE
  end if
elseif menunum = 4 then ' FUNCTIONS menu
  if itemnum = 1 then  ' WINDOWS
    gosub 40          ' Gosub WINDOWS
  elseif itemnum=2 then ' SOUNDER
    gosub 50          ' Gosub SOUNDER
  end if
elseif menunum = 5 then ' SPECIAL menu
  if itemnum = 1 then  ' TEXTSIZE
    gosub 60          ' Gosub TEXTSIZE
  elseif itemnum=2 then ' TEXTCOLOR
    gosub 70          ' Gosub TEXTCOLOR
  elseif itemnum=3 then ' TEXTFACE
    gosub 80          ' Gosub TEXTFACE
  elseif itemnum=4 then ' TEXTFONT
    gosub 90          ' Gosub TEXTFONT
  elseif itemnum=5 then ' TEXTBCOLOR
    gosub 100         ' Gosub TEXTBCOLOR
  end if
elseif menunum = 6 then ' HELP menu
  if itemnum = 1 then  ' HELP FILE

gosub 120            ' Gosub HELP
elseif itemnum=2 then ' ABOUT
gosub 130            ' Gosub ABOUT
end if
end if
return

' -----
' You can use this routine to read text
' files off disk and show them on screen.

10:                  ' READ a file
f$ = files$(1,"TXTSRC") ' File dialog box
if f$="" then goto 15 ' CANCEL clicked
windownum = windownum+1 ' Prepare a window
' The following should be 1 line, no spaces:
window windownum,, (left+25,top+28)
- (right-25,bottom-15),2
open f$ for input as 2 ' Open the file
while not eof(2)
  line input #2, a$
  print a$              ' Print to screen
wend
close #2
print:print spc(30); "<CLICK>"
while not mouse(0) = 0 ' Delay loop
wend
while mouse(0) = 0     ' Await click
wend
window close windownum ' Close the window
windownum = windownum - 1
goto 10
15:
menu                  ' Un-inverse menu
return

' -----
' This procedure reads a text file from
' disk and routes it to the printer:

20:                  ' PRINT a file
f$ = files$(1,"TXTSRC") ' Call file dialog
if f$="" then goto 25 ' CANCEL clicked
open f$ for input as 1
' Open printer as a device:
open "LPT1:PROMPT" for output as 2
while not eof(1)
  line input #1, a$
  print #2, a$         ' Print it
wend
close #2
close #1
goto 20
25:
menu                  ' Un-inverse menu
return

' -----
' This routines deletes files:

30:                  ' DELETE a file
f$ = files$(1)        ' File dialog box
if f$="" then goto 35 ' CANCEL clicked
kill f$
goto 30
35:
menu                  ' Un-inverse menu
return

' -----
' This routine allows you to create windows
' in various sizes, styles and locations.

```

```

\ Only 8 windows are allowed, but Exerciser
\ does not perform error-checking for this:

40:                                \ WINDOWS
windownum = windownum+1           \ Prepare window
\ The window call should be 1 line only:
window windownum,"", (left+25,top+13+15)
                                - (right-25,bottom-15),2
x% = windownum + 1                \ Scratch window #

wind:
print
print "AC/Windows:"
print
input "Enter title          : ";title$
input "Enter leftX (15-640) : ";leftx%
if leftx% = 0 then leftx% = 15
input "Enter topY (15-200)  : ";lefty%
if lefty% = 0 then lefty% = 15
input "Enter rightX (15-640) : ";rightx%
if rightx% = 0 then rightx% = 600
input "Enter bottomY (15-200) : ";righty%
if righty% = 0 then righty% = 175
input "Enter style          : ";style%
if style% = 0 then style% = 1
\ The window call should be 1 line only:
window x%,title$, (leftx%,lefty%)
                                - (rightx%,righty%),style%
x% = x% + 1                      \ the window number
print
print "<RETURN> to continue, <Q> to quit";
input i$
Cmd$=ucase$(i$)                  \ Make input UPPERCASE
if Cmd$="Q" then goto wrtrn
goto wind

wrtrn:                            \ Close windows we created
for t% = x% to windownum step -1
window close t%
next
menu                              \ Un-inverse menu bar
windownum = windownum - 1
return

\ -----
\ Issue single notes, demonstrate REPEAT:

50:                                \ SOUNDER
windownum = windownum+1           \ New window
\ The window call should be 1 line only:
window windownum,"", (left+25,top+13+15)
                                - (right-25,bottom-15),2

sound1:
print
print "AC/Sounder:"
print
\ Interval between notes (seconds):
input "Select interval (xx) : ";i%
if i% = 0 then i% = 90
\ Duration of note in seconds:
input "Select duration (xx) : ";d%
if d% = 0 then d% = 60
\ Low C to High G:
input "Select tone (0-127) : ";t%
if t% = 0 then t% = 66
\ .. a note I like
\ Loudness:
input "Select volume (1-255): ";l%
if l% = 0 then l% = 150
print
print "SOUND ";i%,"";d%,"";t%,"";l%
\ Play and repeat:

sound 3,i%,d%,t%,l%,repeat
print
print "<RETURN> to continue, <Q> to quit";
input i$
Cmd$=ucase$(i$)
if Cmd$="Q" then goto rtrn
sound stop                        \ Stop sound
cls                               \ Clear the screen
goto sound1                      \ And cycle

rtrn:
sound stop
window close windownum
windownum = windownum - 1
menu
return

\ -----
\ Play with font scaling:

60:                                \ TEXT SIZE
windownum = windownum + 1         \ New window
\ The window call should be 1 line only:
window windownum,"", (left+25,top+13+15)
                                - (right-25,bottom-15),2

tsize:
print
print "Text Size:"
print
print "Normal text is 8-point."
print
input "Set size (1-255)      : ";s%
if s% = 0 then s% = 8
if (s%<1 or s%>255) then goto tsize
textsize s%
print
print "Text point size is now ";s%
print
print "<RETURN> to continue, <Q> to quit";
input i$
Cmd$=ucase$(i$)
if Cmd$="Q" then goto trtrn
cls                               \ Clear the screen
goto tsize                      \ And cycle

trtrn:
window close windownum \ Close the window
windownum = windownum - 1
menu
return

\ -----
\ Play with text colors:

70:                                \ TEXT COLOR
windownum = windownum + 1         \ New window
\ The window call should be 1 line only:
window windownum,"", (left+25,top+13+15)
                                - (right-25,bottom-15),2

tcolor:
print
print "Text Color:"
print
print "Default color is 0 (black)."
print "3, White, will be invisible."
print
input "Set color (0-3)       : ";c%
if (c%<0 or c%>3) then goto tcolor
textcolor c%

```

```

print
print "Text color is now #";c%
print
print "<RETURN> to continue, <Q> to quit";
input i$
Cmd$=ucase$(i$)
if Cmd$="Q" then goto crtrn
cls                                     \ Clear the screen
goto tcolor                             \ And cycle

crtrn:
window close windownum                 \ Close the window
windownum = windownum - 1
menu
return

-----
\ Play with font attributes:

80:                                     \ TEXT FACE
windownum = windownum + 1               \ New window
\ The window call should be 1 line only:
window windownum,"", (left+25,top+13+15)
    -(right-25,bottom-15),2

tface:
print
print "Text Face:"
print
print
print "Default face is 0 (Plain Text)."
print
input "Set face (0-16)          : ";f%
if (f%<0 or f%>16) then goto tface
textface f%
print
print "Text face is now #";f%
print
print "<RETURN> to continue, <Q> to quit";
input i$
Cmd$=ucase$(i$)
if Cmd$="Q" then goto frtrn
cls                                     \ Clear the screen
goto tface                             \ And cycle

frtrn:
window close windownum                 \ Close the window
windownum = windownum - 1
menu
return

-----
\ Change to new font:

90:                                     \ TEXT FONT
windownum = windownum + 1               \ New window
\ The window call should be 1 line only:
window windownum,"", (left+25,top+13+15)
    -(right-25,bottom-15),2

tfont:
print
print "Text Font:"
print
print
print "Default font is 0 (Geneva)."
print "Font must exist on system volume."
print
input "Set font (-32768 - 32768): ";tf%
if (tf%<-32768 or tf%>32768) then
    goto tfont
end if
textfont tf%

print
print "Text face is now #";tf%
print
print "<RETURN> to continue, <Q> to quit";
input i$
Cmd$=ucase$(i$)
if Cmd$="Q" then goto tfrtrn
cls                                     \ Clear the screen
goto tfont                             \ And cycle

tfrtrn:
window close windownum                 \ Close the window
windownum = windownum - 1
menu
return

-----
\ Set text background color:

100:                                    \ TEXTBCOLOR
windownum = windownum + 1               \ New window
\ The window call should be 1 line only:
window windownum,"", (left+25,top+13+15)
    -(right-25,bottom-15),2

tbcolor:
print
print "Text Background Color:"
print
print
print "Default color is 3 (White)."
print
input "Set color (0-3)          : ";bc%
if (bc%<0 or bc%>3) then goto tbcolor
textbcolor bc%
print
print "Text background color is now #";bc%
print
print "<RETURN> to continue, <Q> to quit";
input i$
Cmd$=ucase$(i$)
if Cmd$="Q" then goto tbcrtm
cls                                     \ Clear the screen
goto tbcolor                           \ And cycle

tbcrtm:
window close windownum                 \ Close the window
windownum = windownum - 1
menu
return

-----
\ The HELP file:

120:                                    \ HELP FILE
windownum = windownum + 1               \ New window
\ The window call should be 1 line only:
window windownum,"", (left+25,top+13+15)
    -(right-25,bottom-15),2
chdir "1/FW.PROCS/"                    \ Vol for help file
open "AC.EX.HELP" for input as 2       \ Open it
while not eof(2)
    line input #2, a$
    print a$                             \ Print to screen
wend
close #2
print:print spc(30);"<CLICK>"
while not mouse(0) = 0                  \ Delay loop
wend
while mouse(0) = 0
wend
window close windownum                 \ Close the window
windownum = windownum - 1

```

```

menu                                     \ Un-inverse menu
return

-----

\ The special ABOUT box:

130:                                     \ ABOUT
windownum = windownum + 1               \ New window
\ The window call should be 1 line only:
window windownum, "", (left+25,top+13+15)
                                     - (right-25,bottom-15), 2
chdir "1/FW.PROCS/"                     \ About file vol
open "AC.EX.ABOUT" for input as 2 \ Open
while not eof(2)
    line input #2, a$
    print a$                             \ Print to screen
wend
close #2
print:print spc(30); "<CLICK>"
while not mouse(0) = 0                 \ Delay loop
wend
while mouse(0) = 0
wend
window close windownum                 \ Close window
windownum = windownum - 1
menu                                     \ Un-inverse menu
return

-----

\ Customize this loop to set up your desktop
\ menu bars. Menus are number from 1 to x,
\ from the left. (1 & 2 are FILE and EDIT
\ unless you tell the compiler to leave
\ them out):

setmenu:                                \ Setup menu bar
menu 3,0,1,"Files"                      \ Install FILES
menu 3,1,1,"Read"                       \ Read
menu 3,2,1,"Print"                      \ Print
menu 3,3,1,"Delete"                    \ Delete
menu 4,0,1,"Functions"                  \ FUNCTIONS menu
menu 4,1,1,"Windows"                   \ Windows
menu 4,2,1,"Sounder"                   \ Sounder
menu 5,0,1,"Special"                    \ SPECIAL menu
menu 5,1,1,"Text Size"                  \ Text size
menu 5,2,1,"Text Color"                 \ Text color
menu 5,3,1,"Text Face"                  \ Text face
menu 5,4,1,"Text Font"                  \ Text font
menu 5,5,1,"Text BColor"                \ Back color
menu 6,0,1,"Help"                       \ HELP menu
menu 6,1,1,"Help File"                  \ Help
menu 6,2,1,"About ..."                \ About
return

-----

\ Gather all of the definitions etcetera
\ together in one place:

setup:                                   \ Setup program
top = 0                                 \ Screen dimensions
left = 0                                \ This technique
bottom = 199                            \ helps portability
right = 639
windownum = 2                           \ Default window #
return

-----

```

Support File 1

The AC/Exerciser is a program that demonstrates the functionality of AC/BASIC for the Apple IIgs. It should be used in conjunction with the article "Launching Into BASIC — AC/BASIC" in Call -A.P.P.L.E. magazine.

AC/Exerciser is not a traditional demo, instead it lets you issue language calls directly, often with custom parameters. The source code to the program is heavily commented to show how each task is performed.

This program is copyrighted, but may be copied and distributed freely. Written consent is required to include it in any commercial package. Portions of this program include material copyrighted (C) by Absoft Corp. 1988. Used with permission. All other copyrights acknowledged.

Support File 2

AC/Exerciser demonstrates selected capabilities of AC/BASIC. The FILES menu shows how disk files can be read, printed and deleted. FUNCTIONS demonstrates how windows and sound are generated.

Finally, some of AC/BASIC's special text abilities are shown. Each menu selection will ask you for some response. For instance, when you are using WINDOWS, you will be asked to enter the four dimensions of the window. Whenever such information is needed, you can type return to accept the built-in defaults. Be careful not to exceed the suggested defaults. Doing so can cause a system lockup or crash.

Product:

AC/BASIC

Company:

Absoft Corp
2781 Bond Street
Auburn Hills, MI 48057
(313) 853-0050

Price: \$125.00

CIRCLE 20 ON READER SERVICE CARD

Product:

TML BASIC

Company:

TML Systems, Inc
8837-B Goodbys Executive Drive
Jacksonville, FL 32217
(904) 636-8592

Price: \$125.00

CIRCLE 21 ON READER SERVICE CARD

Product:

GSBASIC

Company:

Apple Computer, Inc.
APDA™
290 S.W. 43rd Street
Renton, WA 98055
(206) 251-5222

(APDA is administered by, but is separate from, TechAlliance.)

Price: \$50.00

CIRCLE 22 ON READER SERVICE CARD