



OPEN DIALOG

Desktop dialogs lend
elegance and credibility to
the programs you create.

By JOE ABERNATHY

THE APPLE IIgs DESKTOP HAS A carefully crafted way of communicating with you. It's called the *dialog*. This elegant interface requests the information needed to complete a command, cautions you as to when you're about to make a mistake, or prompts you to fill out an entire data-entry form. In fact, whenever a desktop program communicates with you, there's a dialog for the job.

Dialog boxes are implemented with the help of a software tool set called the *Dialog Manager* (TOOL21), which is built into the IIgs. To use the Dialog Manager, you must first be using a language compiler that supports tool calls. For BASIC, that means one of the three compilers: Micol, TML, and AC/BASIC. Finally, you must see to a number of tool-set interdependencies implied in any program that implements desktop-style features.

For **AC/BASIC**, the accompanying source-code listings will show you everything you need to make your dialogs work. For **TML BASIC**, you must use a desktop program shell of some sort to start the underlying tools, such as the Window Manager, that the Dialog Manager requires. If you don't have such a shell program, you can obtain mine free from *inCider* by sending a stamped (65¢ or appropriate postage for foreign destinations), legal-sized, self-addressed envelope. For **Micol BASIC**, you'll need a shell such as that included

with the compiler beginning with version 3.1.

While it isn't necessary to follow *this* column, a full mastery of dialogs will demand that you buy both volumes of the *Apple IIgs Toolbox Reference*, along with the *Apple Human Interface Guidelines* (Addison-Wesley, Reading, MA). You can find all three references at your bookstore, or via mail from the **Apple Programmers and Developers Association** (APDA, 290 SW 43rd Street, Renton, WA 98055, 206-251-6548). A beta-level revision to the toolbox references that includes GS/OS version 5.0 information is available only through APDA, although this information is not yet germane to any of the BASIC compilers. (Note that Micol and AC/BASIC are compatible with GS/OS version 5.0. TML BASIC is compatible with ProDOS 16 version 3.2.)

WORK SMARTER, NOT HARDER

The Dialog Manager is one of the smartest tool sets in the IIgs, and that helps you look good. The standard dialog features allow text editing; multiple-choice buttons; check boxes, which let you customize program operation; and radio buttons, which let you select among a group of options, such as paper size for printing. Dialogs may contain icons such as the ones the Finder uses, or entire picture images; they may contain custom controls, such as a thermometer that monitors the progress of a disk operation.

The Dialog Manager can produce three

families of dialog windows: modal dialogs, modeless dialogs, and alerts. *Modal dialogs* are those that demand your input, such as "Information is about to be erased. Continue?" *Modeless dialogs* are those that hang out in the background, like a spelling checker, waiting to be of use. *Alerts* are communications between you and the program, such as "Please wait for a time-consuming disk operation."

Like most Toolbox features, dialogs require that the programmer supply a list of information defining what the dialogs should do, and when and how. If you're implementing a simple dialog for something like the "About this program" window, almost everything in the list will have a default: You won't have to do much work. With each level of sophistication you add, however, things get more complex from a programmer's standpoint, so that you might eventually find yourself managing hundreds of lines of code for a sophisticated game screen implemented via the Dialog Manager.

The list of information a dialog requires is called a *data structure*, and it highlights the overriding weakness of all the existing BASIC compilers. BASIC just wasn't built for sophisticated data structures; it was built for strings and arrays. The people who wrote the modern compilers, not wanting to lose the flavor of BASIC, haven't seen fit to provide an extended set of data structures. Rather, each offers its own quirky interface to the Toolbox. So we end up with tradeoffs.

Throughout all our BASICs—in every language compiler—there are tradeoffs that create a bit more work for you than otherwise might be the case. Don't let yourself become discouraged. If you're trying to master dialogs or any other desktop programming task, jot down the structure of the "pure" call as it appears in the toolbox references; examine any similar source code you can find, no matter its language; consult your compiler ►

manual, particularly in regard to its handling of data structures; and make it work.

BRING IT TO LIFE

Each BASIC compiler makes you walk a different path in getting dialogs up and running, but there are more similarities than it would seem. To illustrate this column, I wanted to create a dialog through which you might build a mailing list respectable enough to even fit into business-quality applications. What do we need to do this? A set of very predictable actions, it turns out:

- Prompt the user for input.
- Get input. In the case of a dialog, that means allowing the manipulation of text, and of various types of buttons.
- Analyze and act on the input, or determine the final state of each item in a dialog box and react accordingly.

To build a dialog that does this, you'd do the following:

- Create a new dialog, adding each item individually. Display the dialog.
- Write a loop to handle the dialog's logic. In the case of a one-button ("OK") dialog, your logic would just wait for any event to occur. Once anything transpires, you know OK was chosen.

In more advanced dialogs, you'd check for action in specific item types and specific items and react accordingly. If you have four radio buttons and click on the third one, you must deselect the other three and make sure the one that took the dialog hit is selected. Check boxes must be turned on and off. You can check for actions in standard pushbuttons, and depending on the buttons you create, initiate a suitable response. As for EditLine text items, Dialog Manager handles all the details for you—unless you use AC/BASIC, which makes you monitor mouse clicks and TAB keys.

- Determine the results of your interaction with the dialog. Extract the EditLine strings into usable form; set a flag showing which of the radio buttons was chosen; set individual flags for each check item to determine whether it was turned on or off. Handle standard pushbuttons as needed; if you create an "OK" button with an itemID of 1, Dialog Manager will consider it the default and accept *Return* as though it were a mouse click on the OK button.

TML BASIC, despite being incompatible with IIGS system software published since 1987, stays the closest to the intent of the Dialog Manager as described in the toolbox references. So let's examine the logic for managing our mailing-list dialog in TML. To do so, look ahead to **Listing 1**, directly below the `_NewDItem` calls, at the `DO/UNTIL` loop.

First, an exit flag is set to empty. The first checks made thereafter are for clicks on the OK or Cancel buttons; either of these events sets *exit%*, which makes the control loop exit.

If you get past *exit%*, you know you've done something in one of the other fields, so check to see which type of dialog item took the hit. If it was a radio item, store that particular item in a variable, so that you can use the information later; then cycle the status of the radio buttons to make sure only one is selected.

If your item type was "check item" instead of "radio," check to see whether the check item is currently on or off, then switch it, because a click means you want it to be in the only other state available. In the example dialog, the "Preferred%" variable will be set if the customer in question is a preferred customer, or set to zero otherwise.

Having made it through the entire logic loop, that takes care of everything except determining what was typed in the EditLine items.

Listing 1. TML BASIC mailing-list dialog.

```
' This source code implements a mailing list dialog suitable for use
' in business applications. It includes the necessary logic to let a user
' interact with the dialog features; and the logic to extract the results
' of the user's interaction, including EditLine text items, check boxes,
' radio button items and standard buttons.

' If you are using my TML starter shell, you can add the address dialog to
' your shell with the following source code:

' Add "DoAddrDialog" to the SetUpMenus procedure:

PROC SetUpMenus          ' Build the menus and menu bar
...

MENUDEF 16,DoSound       ' Play a sound file
MENUDEF 17,DoAddrDialog  ' Address list dialog

...

ENDPROC

' Add this label to the shell:

DoAddrDialog:
PROC AddressDialog       ' Requires DESKTOOLS library
RETURN 0

' Change the "GoodiesMenu" procedure in the file DESKTOOLS
' so that it reads like this:

DEF PROC GoodiesMenu     ' Create goodies menu
LOCAL MenuStr$
MenuStr$ = "    >>> Goodies \N5\0"
MenuStr$ = MenuStr$ + "==Play Sound\N266\0"
MenuStr$ = MenuStr$ + "==Address Dialog\N267\0"
SET(GoodMenuStr$(0)) = "MenuStr$
InsertMenu(EXFN_NewMenu(VARPTR(GoodMenuStr$(1))),0)
END PROC GoodiesMenu

' Add this procedure, which does the actual work, to DESKTOOLS:

-----
' Address Dialog. Displays a dialog to allow entry of information for
' an address list, then retrieves the information entered. Demonstrates
' how to use all of the standard dialog items -- buttons, radio items,
' check boxes, edit lines.

DEF PROC AddressDialog
LOCAL Dialog$,itemHit$,MyStr$,CancelStr$,Item1$,Item2$,Item3$,Item4$
LOCAL Item5$,Item6$,Item7$,Item8$,Item9$,Item10$,Item11$,Item12$
LOCAL Item13$,Item14$,Item15$,Item16$,Item17$,Item18$,Item19$,Item20$
LOCAL Item21$,Item22$,Item23$,Item24$,Name$,Title$,Company$,Street$
LOCAL City$,State$,Zip$,Nation$,AreaCode$,Phone$,Sales$,Preferred$
' The dialog window rectangle:
SetRect(VARPTR(aRect$(0)),0,18,632,195)
' Get a handle to the dialog:
Dialog$ = EXFN_NewModalDialog(VARPTR(aRect$(0)),1,0)
SetForeColor(0)      ' black letters on
SetBackColor(15)     ' white background

' Create each item in the dialog. The position of each item is determined
' by the four parameters given in the _SetRect call:

Item24$ = "Sales to Date"
SetRect(VARPTR(aRect$(0)),354,34,450,43)
_NewDItem(Dialog$,26,VARPTR(aRect$(0)),15,VARPTR$(Item24$),0,0,0)

Item23$ = "Phone"
SetRect(VARPTR(aRect$(0)),359,14,399,23)
_NewDItem(Dialog$,25,VARPTR(aRect$(0)),15,VARPTR$(Item23$),0,0,0)

Item22$ = "Nation"
SetRect(VARPTR(aRect$(0)),20,117,66,126)
_NewDItem(Dialog$,24,VARPTR(aRect$(0)),15,VARPTR$(Item22$),0,0,0)

Item21$ = "State"
SetRect(VARPTR(aRect$(0)),241,96,281,105)
_NewDItem(Dialog$,23,VARPTR(aRect$(0)),15,VARPTR$(Item21$),0,0,0)

Item20$ = "City"
SetRect(VARPTR(aRect$(0)),34,96,64,105)
_NewDItem(Dialog$,22,VARPTR(aRect$(0)),15,VARPTR$(Item20$),0,0,0)

Item19$ = "Street"
SetRect(VARPTR(aRect$(0)),19,75,66,84)
_NewDItem(Dialog$,21,VARPTR(aRect$(0)),15,VARPTR$(Item19$),0,0,0)

Item18$ = "Company"
SetRect(VARPTR(aRect$(0)),11,54,68,63)
_NewDItem(Dialog$,20,VARPTR(aRect$(0)),15,VARPTR$(Item18$),0,0,0)

Item17$ = "Title"
SetRect(VARPTR(aRect$(0)),28,33,66,42)
_NewDItem(Dialog$,19,VARPTR(aRect$(0)),15,VARPTR$(Item17$),0,0,0)

Item16$ = "Name"
SetRect(VARPTR(aRect$(0)),33,13,66,22)
_NewDItem(Dialog$,18,VARPTR(aRect$(0)),15,VARPTR$(Item16$),0,0,0)

Item15$ = "Joe Abernathy"
SetRect(VARPTR(aRect$(0)),73,10,319,23)
_NewDItem(Dialog$,17,VARPTR(aRect$(0)),17,VARPTR$(Item15$),20,0,0)

Item14$ = "526-9711"
SetRect(VARPTR(aRect$(0)),459,11,561,24)
_NewDItem(Dialog$,16,VARPTR(aRect$(0)),17,VARPTR$(Item14$),8,0,0)

Item13$ = "713"
SetRect(VARPTR(aRect$(0)),404,11,446,24)
_NewDItem(Dialog$,15,VARPTR(aRect$(0)),17,VARPTR$(Item13$),3,0,0)

Item12$ = "77266-6046"
SetRect(VARPTR(aRect$(0)),216,114,318,127)
_NewDItem(Dialog$,14,VARPTR(aRect$(0)),17,VARPTR$(Item12$),10,0,0)

Item11$ = "U.S."
SetRect(VARPTR(aRect$(0)),71,114,197,127)
_NewDItem(Dialog$,13,VARPTR(aRect$(0)),17,VARPTR$(Item11$),10,0,0)
```

Continued

AT LAST!



McGEE... A VERY SPECIAL PROGRAM FOR YOUR PRESCHOOLER

A totally new concept
in preschool software...
McGEE is a program with

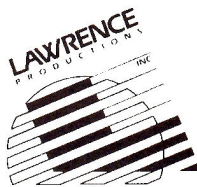
**NO
WORDS!**

Your kids don't have to be able to read to explore McGEE's "house" and decide what they want to do. Things like bounce the ball, give the dog a treat, ride the hobby horse, swing in the back yard and much, much more. Bright and sparkling graphics with realistic audio make McGEE fun and easy to play.

With McGEE, you're giving your children more than just a fun game to play. You're helping them develop the computer sense they'll need to compete in tomorrow's world.

Get your kid off to a head start with this truly exceptional new program — only from Lawrence Productions, Inc., a national leader in the development of educational software products.

Order your copy of McGEE today. Specify Apple IIGs or Macintosh version. MasterCard and Visa accepted.



**LAWRENCE
PRODUCTIONS INC.**
Department G-54
1800 35th Street
Galesburg, MI 49053
800-421-4157

\$39.95

Plus Shipping & Handling

APPLE IIGs BASICS

Continued

```
Item10$ = "TX"
SetRect (VARPTR(aRect$(0)), 287, 93, 317, 106)
NewDItem (Dialog@, 12, VARPTR(aRect$(0)), 17, VARPTR$(Item10$), 2, 0, 0)

Item9$ = "Houston"
SetRect (VARPTR(aRect$(0)), 71, 93, 221, 106)
NewDItem (Dialog@, 11, VARPTR(aRect$(0)), 17, VARPTR$(Item9$), 12, 0, 0)

Item8$ = "P.O. Box 66046"
SetRect (VARPTR(aRect$(0)), 71, 72, 317, 85)
NewDItem (Dialog@, 10, VARPTR(aRect$(0)), 17, VARPTR$(Item8$), 20, 0, 0)

Item7$ = "First Word"
SetRect (VARPTR(aRect$(0)), 72, 51, 318, 64)
NewDItem (Dialog@, 9, VARPTR(aRect$(0)), 17, VARPTR$(Item7$), 20, 0, 0)

Item6$ = "Tech Support"
SetRect (VARPTR(aRect$(0)), 73, 30, 319, 43)
NewDItem (Dialog@, 8, VARPTR(aRect$(0)), 17, VARPTR$(Item6$), 20, 0, 0)

Item5$ = "Over 2000"
SetRect (VARPTR(aRect$(0)), 401, 96, 494, 185)
NewDItem (Dialog@, 7, VARPTR(aRect$(0)), 12, VARPTR$(Item5$), 0, 0, 0)

Item4$ = "1000-2000"
SetRect (VARPTR(aRect$(0)), 401, 80, 498, 89)
NewDItem (Dialog@, 6, VARPTR(aRect$(0)), 12, VARPTR$(Item4$), 0, 0, 0)

Item3$ = "500-1000"
SetRect (VARPTR(aRect$(0)), 401, 64, 490, 73)
NewDItem (Dialog@, 5, VARPTR(aRect$(0)), 12, VARPTR$(Item3$), 0, 0, 0)

Item2$ = "0-500"
SetRect (VARPTR(aRect$(0)), 401, 49, 465, 58)
NewDItem (Dialog@, 4, VARPTR(aRect$(0)), 12, VARPTR$(Item2$), 1, 0, 0)

Item1$ = "Preferred Customer"
SetRect (VARPTR(aRect$(0)), 353, 116, 519, 125)
NewDItem (Dialog@, 3, VARPTR(aRect$(0)), 11, VARPTR$(Item1$), 0, 0, 0)

CancelStr$ = "Cancel"
SetRect (VARPTR(aRect$(0)), 230, 142, 308, 156)
NewDItem (Dialog@, 2, VARPTR(aRect$(0)), 10, VARPTR$(CancelStr$), 0, 0, 0)

' Any dialog item given an item reference number of one (in the
' NewDItem call) will be the default item, meaning the user can
' press return to activate the default. In most cases, item 1
' should be the OK button.

OKStr$ = "OK"
SetRect (VARPTR(aRect$(0)), 344, 142, 391, 156)
NewDItem (Dialog@, 1, VARPTR(aRect$(0)), 10, VARPTR$(OKStr$), 0, 0, 0)

' Dialogs that do anything more than wait for a default OK event
' must supply the logic for dealing with the various dialog items.
' This is shown below:

' Process events until OK or Cancel pressed:
exit% = 0 ' Set by OK button
DO
  itemHit% = EXPN ModalDialog@
  IF itemHit% = 1 THEN exit% = 1 ' OK
  IF itemHit% = 2 THEN exit% = 1 ' Cancel
  myhit% = EXPN GetDItem@ (Dialog@, itemHit%)
  IF myhit% = 12 THEN ' it was a radio item
    Sales% = itemHit% ' store for later use
    SetDItemValue (1, Dialog@, itemHit%)
    ' Cycle the radio buttons on and off:
    IF itemHit% <> 4 THEN
      SetDItemValue (0, Dialog@, 4)
    END IF
    IF itemHit% <> 5 THEN
      SetDItemValue (0, Dialog@, 5)
    END IF
    IF itemHit% <> 6 THEN
      SetDItemValue (0, Dialog@, 6)
    END IF
    IF itemHit% <> 7 THEN
      SetDItemValue (0, Dialog@, 7)
    END IF
  END IF
  IF myhit% = 11 THEN ' check item
    myval% = EXPN GetDItemValue (Dialog@, itemHit%)
    IF myval% = 0 THEN
      SetDItemValue (1, Dialog@, itemHit%)
      Preferred% = 1 ' store for later use
    ELSE
      SetDItemValue (0, Dialog@, itemHit%)
      Preferred% = 0 ' store for later use
    END IF
  END IF
UNTIL exit% = 1

' Extract strings from EditLine items. To use these, you can
' do a simple "PRINT Name$", etc. (Note that these values must be
' assigned to global variables to be used outside this procedure:)
GetIText* (Dialog@, 17, VARPTR$(Name$))
GetIText* (Dialog@, 16, VARPTR$(Phone$))
GetIText* (Dialog@, 15, VARPTR$(AreaCode$))
GetIText* (Dialog@, 14, VARPTR$(Zip$))
GetIText* (Dialog@, 13, VARPTR$(Nation$))
GetIText* (Dialog@, 12, VARPTR$(State$))
GetIText* (Dialog@, 11, VARPTR$(City$))
GetIText* (Dialog@, 10, VARPTR$(Street$))
GetIText* (Dialog@, 9, VARPTR$(Company$))
GetIText* (Dialog@, 8, VARPTR$(Title$))

' All done:
CloseDialog (Dialog@)
END PROC AddressDialog
```


WYSIWYG Editors

Something that'll simplify programming GS desktop elements such as dialogs is worth its weight in gold. Two recent products sporting WYSIWYG (what you see is what you get) desktop editors deserve platinum: *Design Master* (Byte-Works, 4700 Irving Boulevard Northwest, Suite 207, Albuquerque, NM 87114, 505-898-8183) and *CallBox TPS* (So What Software, 10221 Slater Avenue, Suite 103, Fountain Valley, CA 92708, 714-964-4298, \$99). With WYSIWYG tools, creating and editing Apple's Human Interface Guidelines desktop simply involves selecting desktop elements from pull-down menus and, using a mouse, placing them onto the 320- or 640-graphics desktop with clicks and drags.

Chris Haun promotes his creation, *Design Master*, as a desktop "prototyping" tool. The results are source-code compatible with most of the 65816 assemblers and advanced languages available for the GS, including C, Pascal, and BASIC.

CallBox TPS also includes a WYSIWYG image editor for the creation of icons, cursors, and pixel elements for 320- and 640-mode super-high-resolution GS graphics screens, plus an Applesoft BASIC interface package.

Now even the novice programmer can personalize his or her desktop environments quickly and easily. □

—Bill Kennedy, Technical Editor

The _GetIText tool call handles that; it extracts the EditLine information into strings usable in any traditional fashion for which you'd use a string.

That's it for the logic, and it's really pretty simple programming for something that can do so much. The differences among the three compilers are minor, and notes in the source-code listings alert you to the differences that do exist. (See Listings 2 and 3.) Note that at the time I wrote this column, the new version 3.1 of Microl BASIC supporting direct desktop programming was still in beta release, and wouldn't support the sample address dialog. You'll find an "About..." dialog—which works—in its place by way of example.

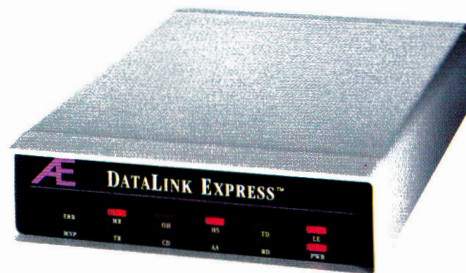
The source-code listings, save that for Microl, show how to implement every standard dialog item. By looking over these examples for just a few moments, you should be able to use them as a basis for creating any standard dialog of low- to medium-level complexity. You won't outgrow them until you reach the point at which you want to define custom procedures for updating the appearance of custom dialog items. And at that point, you won't need me.

A BETTER MOUSETRAP

The biggest drawback to using dialogs is designing their layout. The IIGS screen is based on a grid of either 320 by 200 pixels or 640 by 200 pixels. A sophisticated dialog, such as those in my examples, might have 20 or 30 elements of varying sizes. Now where, with 64,000 or 128,000 screen positions, does each element go? The Dialog Manager isn't smart enough to decide that for you. What you have to do, then, is design each dialog in its entirety before you write a single line of code.

First, there's graph paper. The best you'll get is a 64-by-64 grid, good

DataLink Express™



The Upgradeable External

Waited long enough for an upgradeable, full-featured modem? Introducing the new DataLink Express™ from Applied Engineering. With the *first* comprehensive status light array. The *first* Line Engage indicator. The *first* upgradeable design allowing for the addition of send-Fax capability and MNP error correction. And the *first* to offer these features affordably.

DataLink Express' exclusive Line Engage light indicates whether the phone line is free or in use, before you log on, to help line-sharing users save transmissions from time-wasting interruptions.

DataLink Express incorporates a perfectly matched Apple-platinum case, along with both Apple-type serial port input (Mini-8) and a DB-25 connector for use with PCs. It's fully Hayes compatible and operates at 300, 1200 or 2400 baud. DataLink Express even has non-volatile configuration memory with synchronous and asynchronous communication and separate line and phone connectors for line sharing.

Upgradeability

With our optional send-only Fax (available soon), text and graphics can be composed on your Apple II and directly faxed at 4800 baud. And for the ultimate in data reliability, an MNP option assures 100% accuracy, even if phone lines or other equipment are at fault.

Software, too.

Unlike other modems, you don't have to shop for separate software. Comprehensive communications software for Apple II, Macintosh and MS-DOS computers is included.

Made by the Apple enhancement experts

Best of all, DataLink Express was designed and built by Applied Engineering, long the leader in Apple enhancement products. AE brings ten years' experience to producing feature-laden peripherals that set the standard for quality and reliability.

DataLink Express modem\$249
MNP option\$89

Order today!

To order or for more information, see your dealer or call (214) 241-6060 today, 9 am to 11 pm, 7 days. Or send check or money order to Applied Engineering, MasterCard, VISA and C.O.D. welcome. Texas residents add 7% sales tax. Add \$10 outside U.S.A.

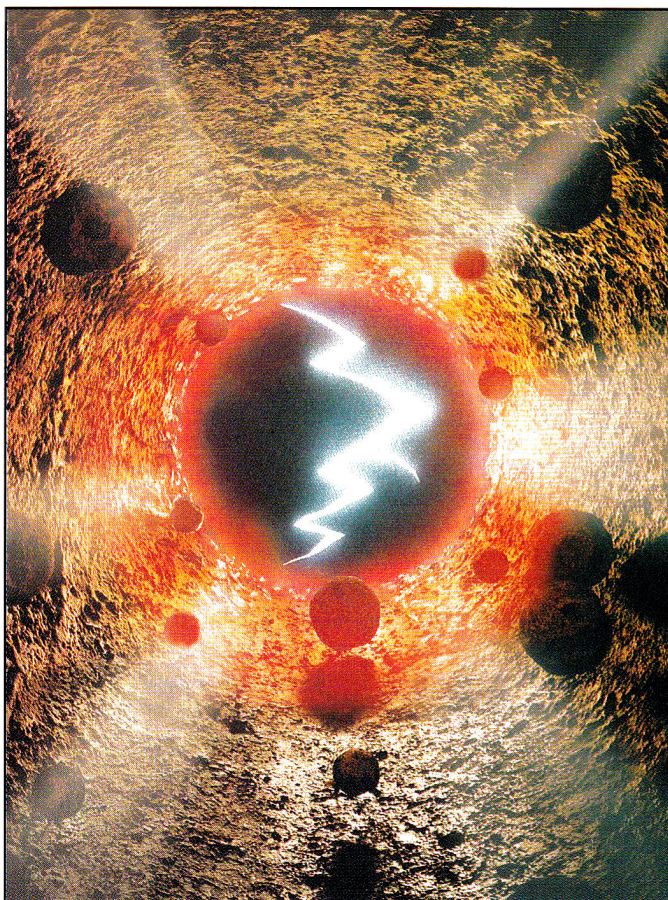
AE APPLIED ENGINEERING®
The Apple enhancement experts.

A Division of AE Research Corporation.

(214) 241-6060

P.O. Box 5100, Carrollton, TX 75011

Prices subject to change without notice. Brand and product names are registered trademarks of their respective holders.



Make Sure You Don't See The Light At The End Of The Tunnel.



Or the future of the world won't be too bright.

You'll be busy swerving past moving barriers, blasting robot guardians, and keeping the walls from caving in on you, not to mention trying to find the doomsday device in time.

Some say you're the best rocket jockey in the galaxy, but are you ready for the TUNNELS OF ARMAGEDDON™? Strap your heart in as you warp through an extensive and deadly network of underground tunnels at mind-numbing speed in your quest to save the world. Pick up special items to help you get past the "nastier" obstacles that await you in the 20 increasingly difficult stages of tunnels. And remember to take the right route, because every second counts. Contact your local dealer for details. Available on Apple IIGS Computers.

TUNNELS OF ARMAGEDDON

The Future Is In Your Hands

"Entertainment Software"

California Dreams
780 Montague Expwy., #403
San Jose, CA 95131
© 1989 Logical Design Works, Inc.

CALIFORNIA
Dreams™

APPLE IIGS BASICS

Listing 2. Micol Advanced BASIC dialog sample.

```
' This source code implements an "About ..." this program dialog. Because
' the version of Micol BASIC supporting desktop programs was still in beta
' release at the time of this writing, it was not possible to implement the
' mailing list dialog.
' This source code is designed to be called from a Micol-style desktop
' menu such as those used in the examples on the Micol v3.0 and later disks.

' Add these data structures at the top of your program:

DIM Array (100)           { dialog data structure array }
DIM Array$ (100)          { dialog text strings array }

{ ----- }
{ About Dialog }
{ Create standard "About ..." dialog box. }
{ Based on an example by Ron Lewin. }

PROC About Dialog [ Control ]
  Array (0) = Control
  CASE OF Control
    DO 0 { Close the Dialog }
      DIALOG (Array (, Array$ ( ) )
    ENDDO
    DO 1 { Create new Dialog Box }

    { First, you feed all of the text items into a text array: }
    Array$ (1) = "Micol BASIC Program Shell"
    Array$ (2) = "By Joe Abernathy"
    Array$ (3) = "OK"

    { Now, you feed all of the attributes of each dialog item into
    { an array. Compare this method with that used by TML. Although
    { Micol's method is easier to understand, it takes about four
    { times as much typing: }
    Array (1) = 30 { Position of Dialog Box, Y Minimum }
    Array (2) = 70 { X Minimum }
    Array (3) = 100 { Y Maximum }
    Array (4) = 500 { X Maximum }
    Array (5) = 3 { Dialog Box contains x Items }
    Array (6) = 1 { This Item Reference #, default }
    Array (7) = 10 { Y Min of Item }
    Array (8) = 10 { X Min of Item }
    Array (9) = 25 { Y Max of Item }
    Array (10) = 400 { X Max of Item }
    Array (11) = 15 { Item type }
    Array (12) = 0 { Enable this Item }
    Array (13) = 0 { Item value, your own }
    Array (14) = 1 { String element 1 }
    Array (15) = 2 { This Item Reference, any unique value }
    Array (16) = 30 { Y Min of Item }
    Array (17) = 10 { X Min of Item }
    Array (18) = 45 { Y Max of Item }
    Array (19) = 400 { X Max of Item }
    Array (20) = 15 { Item type, text }
    Array (21) = 0 { Highlight Item }
    Array (22) = 0 { Item Status }
    Array (23) = 2 { String element 2 }
    Array (24) = 3 { Item ref. }
    Array (25) = 50 { y min }
    Array (26) = 10 { x min }
    Array (27) = 65 { y max }
    Array (28) = 100 { x max }
    Array (29) = 10 { item type, button }
    Array (30) = 0 { item enabled }
    Array (31) = 0 { }
    Array (32) = 3 { string 3 }
    { Display the dialog: }
    DIALOG (Array (, Array$ ( ) )
    MOUSE (Array ( ) { Monitor Response to Dialog }
    { When we get here, an item hit has occurred. First, get
    { a handle to the dialog for use by other tool calls. For
    { this simple dialog, the handle isn't used, but it would
    { be by most dialogs: }
    Dialog_LSB% = Array (0) { Keep Dialog Pointer for TOOLBOX call }
    Dialog_MSB% = Array (1)

    { Now see which item took a hit: }
    Item% = Array (0)

    { There's only one item, so we know the OK button took the hit.
    { Hence, we do nothing but shut things down and exit: }

  ENDDO
ENDCASE
ENDPROC { About Dialog }

{ ----- }
{ DoAbout }
{ Build "About Shell" Dialog Box. }

{ This activates the dialog, then closes it: }
PROC DoAbout
  GOSUB About Dialog [1] { Create }
  { .. and wait for OK button to be clicked }
  GOSUB About Dialog [0] { Close }
ENDPROC { DoAbout }

{ ----- }
{ Main }
{ Program execution occurs here. }

{ This shows how you would implement the logic
{ for a main control loop in Micol, and act on
{ a mouse selection in the "About ..." item: }

PROC MenuTask { heartbeat loop }
  REPEAT
    MOUSE (Array ( )
    Task_Value = Array (0)
    UNTIL Task_Value = 17
    Menu_Item = Array (9)
    Menu_Number = Array (10)
    GOSUB Do Menu [2] { Highlight Menu }
    CASE OF Menu_Item
      DO 256 { About }
        GOSUB DoAbout
```

Continued

Continued

```

      ENDDO
DO 257      { Open }
      { ... }
      ENDDO
DO 258      { Close }
      { ... }
      ENDDO
      { ..... }

      ENDCASE
ENDPROC { MenuTask }

{ This master routine calls the above menu interpreter,
  which handles actual program flow: }

ROUTINE Main
  HGR2      { 640x200 graphics }
  GOSUB SetUp { start other tools }
  GOSUB Do Menu [1] { set up menus }
  GOSUB Do Menu [7] { allow NDAs }
  TOOLBOX {27, 21: 0, 270, 4} { FixFontMenu }
  MOUSE (Array ()) { show menus }
  REPEAT
    GOSUB MenuTask
  UNTIL Done! { Done! is set by QUIT item }
  GOSUB ShutDown { shut down tools }
END { Micol Shell }

```

Listing 3. AC/BASIC mailing-list dialog.

```

' This source code implements a mailing list dialog suitable for use
' in business applications. It includes the necessary logic to let a user
' interact with the dialog features; and the logic to extract the results
' of the user's interaction, including EditLine text items, check boxes,
' radio button items and standard buttons.

' The routine in your program that monitors events in pull-down menus
' must contain routing to the address dialog. If you are using my
' incider AC/BASIC utility/program shell, your menu control procedure
' should look like this:

menuproc:
  menunum = MENU(0)
  itemnum = MENU(1)
  IF menunum = 1 THEN
    IF itemnum = 1 THEN
      GOSUB 10
    ELSEIF itemnum = 2 THEN
      GOSUB 20
    ELSEIF itemnum = 3 THEN
      GOSUB 30
    ELSEIF itemnum = 4 THEN
      GOSUB 40
    ELSEIF itemnum = 5 THEN
      GOSUB 50
    ELSEIF itemnum = 6 THEN
      GOSUB 60
    END IF
  ELSEIF menunum = 2 THEN
    IF itemnum = 1 THEN
      GOSUB 70
    ELSEIF itemnum = 2 THEN
      GOSUB 80
    ELSEIF itemnum = 3 THEN
      GOSUB 90
    END IF
  END IF
  RETURN

' Interpret menu events
' Read which menu
' Read which item
' .. FILE menu
' New
' Edit
' Delete
' Print
' Type File
' Quit
' GOODIES menu
' Show picture
' Play a sound file
' Address dialog

SUB DoMenu
  FOR p = 1 TO 6
    FOR e = 0 TO 12 STEP 4
      PALETTE p,e+0,1,1,1
    NEXT
  NEXT
  MENU 1,0,1,"File"
  MENU 1,1,1,"New"
  MENU 1,2,1,"Edit"
  MENU 1,3,1,"Delete"
  MENU 1,4,1,"Print"
  MENU 1,5,1,"Type"
  MENU 1,6,1,"Quit"
  MENU 2,0,1,"Goodies"
  MENU 2,1,1,"View Picture"
  MENU 2,2,1,"Play Sound"
  MENU 2,3,1,"Address Dialog"
  FOR p = 1 TO 6
    FOR e = 0 TO 12 STEP 4
      PALETTE p,e+0,0,0,0
    NEXT
  NEXT
END SUB

' Create menu bar
' Eliminate screen flicker
' by whitening-out the menu bar
' before building it.
' Thanks to Lee Rimar of
' Absoft for the code.
' Build FILE menu
' and its entries ...

' Goodies menu header
' View SHR picture
' Play a sound file
' Get address info
' Restore original palette ..

' This is the actual dialog work procedure:

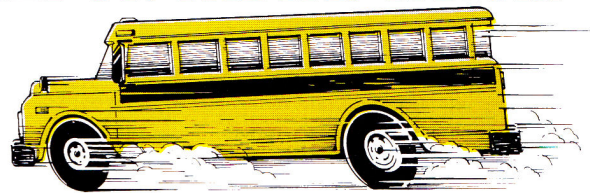
' Address Dialog. Displays a dialog to allow entry of information for
' an address list, then retrieves the information entered. Demonstrates
' how to use all of the standard dialog items -- buttons, radio items,
' check boxes, edit lines.

90:
WINDOW 2,"", (8,18)-(632,195),2
EDIT FIELD 1,"Joe Abernathy", (73,18)-(319,23),1
EDIT FIELD 2,"526-9711", (459,11)-(561,24),1
EDIT FIELD 3,"713", (404,11)-(446,24),1

```

Continued

GET ON THE FAS-TRACK!



Call us for your FREE 1989 Best Sellers Catalog—64 pages of software, hardware and accessories.

DISCOUNTS UP TO 45% EVERYDAY!

Productivity Software

AppleWorks 3.0	\$169.95
AppleWorks GS 1.1	\$199.95
Copy II Plus	\$22.95
HyperStudio 2.0	\$84.50



TimeOut
Series
Starter Packs
Style
\$74.95
Decision
\$84.95

Applied Engineering

GS Ram Plus w/1 Meg	\$239.95
RamKeeper	\$149.95
Vulcan 20 Meg Hard Drive	\$499.95
Vulcan 40 Meg Hard Drive	\$649.95
TransWarp IIgs	\$289.95
TransWarp	\$137.95
RamWorks III w/256K	\$149.95

Performance Starter Pack	\$64.95
BeagleWrite GS (Multiscribe GS)	\$57.95
Superfonts or Telecomm	\$37.50
GS Font Editor or Program Writer	\$28.45
Thesaurus or Powerpack	ea \$27.95
Desktools I or II	ea \$27.95
Sidespread or File Master	ea \$27.95
Spreadtools or Ultramacros	ea \$34.45
Graph or ReportWriter	ea \$42.95

Desktop Publishing & Graphics

Publish It! 2	\$74.95
Childrens' Writing and Pub Center	\$36.95
Print Shop	\$28.95
Print Shop IIgs	\$36.95
Pow Zap Ker-Plunk Comic Maker	\$29.95
VCR Companion	\$29.95
Print Shop Lovers Utility Set	\$25.95
Labels, Labels, Labels	\$25.95

AMR AS800 3.5" Drive



\$189.95
IIgs & IIc+
Daisy-Chainable

Other Hardware

Apple IIe 80 Column 64K Card	\$24.95
FingerPrint GSI Ver 3	\$92.95
4 Mhz Zip Chip	\$123.95
8 Mhz Zip Chip	\$159.95

Prometheus Promodem 2400A



\$129.95
2400 Baud
Internal Modem
for II+, IIe or IIgs.
Includes Software!

256K Drums (set of 8)	\$35
1 Meg Drums (set of 8)	\$109

Education

Where in Time is C. Sandiego	\$27.95
Where in the World is C. Sandiego	\$25.95
Where in the USA is C. Sandiego	\$28.95
Math Blaster or Word Attack Plus	\$29.95
Math Blaster Mystery	\$29.95
Oregon Trail	\$25.95
Think Quick	\$30.95
McGee (IIgs)	\$27.95
Math Rabbit or Reader Rabbit	ea \$26.95
Talking Reader Rabbit (IIgs)	\$35.95
Writer Rabbit	\$29.95
Mavis Beacon Typing (IIgs)	\$33.95

Entertainment

Test Drive II—The Dual (IIgs)	\$26.95
Jack Nicklaus Golf (IIgs)	\$29.95
Grand Prix (IIgs)	\$27.95
Life and Death (IIgs)	\$33.95
Keef the Thief (IIgs)	\$33.95
Prince of Persia	\$23.95
Down Hill Challenge (IIgs)	\$20.95
Chessmaster 2100 (IIgs)	ea \$33.95
Arkanoïd II Return of DOH (IIgs)	\$21.50
Battle Chess (IIgs)	\$29.95
Third Courier (IIgs)	\$30.95
Blue Angels (IIgs)	\$30.95

Accessories

ImageWriter Black Ribbon	\$2.50
ImageWriter 4-Color Ribbon	\$5.75
10 DS/DD 3.5" Bulk Diskettes	\$7.95
25 DS/DD 5.25" Bulk Diskettes	\$8.95
5.25" Disk Case (Holds 60)	\$6.95
3.5" Disk Case (Holds 40)	\$6.95
System Saver IIgs	\$69.95



7030C Huntley Road • Columbus, Ohio 43229

ALWAYS CALL 1-800-272-1600

TOLL-FREE

1-800-438-1168 (Ohio)

1-614-847-4050 (Central Ohio)



U.S., F.P.O., and A.P.O., add 3% (minimum \$4.00) for each shipment. No C.O.D. In Ohio, add 5.5% Sales Tax. MasterCard, VISA, and American Express—No extra charge. We accept purchase orders from schools, universities and other qualified organizations.

FROM HOME TO SCHOOL AND BACK AGAIN, FAS-TRACK DELIVERS!

Continued

```

EDIT FIELD 4,"77266-6046",(216,114)-(318,127),1
EDIT FIELD 5,"U.S.",(71,114)-(197,127),1
EDIT FIELD 6,"TX",(287,93)-(317,106),1
EDIT FIELD 7,"Houston",(71,93)-(221,106),1
EDIT FIELD 8,"P.O. Box 66046",(71,72)-(317,85),1
EDIT FIELD 9,"First Word",(72,51)-(318,64),1
EDIT FIELD 10,"Tech Support",(73,30)-(319,43),1
BUTTON 7,1,"Over 2000",(401,96)-(494,105),3
BUTTON 6,1,"1000-2000",(401,80)-(496,89),3
BUTTON 5,1,"500-1000",(401,64)-(490,73),3
BUTTON 4,2,"0-500",(401,49)-(465,58),3
BUTTON 3,1,"Preferred Customer",(353,116)-(519,125),2
BUTTON 2,1,"Cancel",(230,142)-(308,156),1
BUTTON 1,1,"OK",(344,142)-(391,156),1
LOCATE 5,42
PRINT "Sales to Date"
LOCATE 3,42
PRINT "Phone"
LOCATE 14,2
PRINT "Nation"
LOCATE 12,30
PRINT "State"
LOCATE 12,2
PRINT "City"
LOCATE 9,2
PRINT "Street"
LOCATE 7,2
PRINT "Company"
LOCATE 5,2
PRINT "Title"
LOCATE 3,2
PRINT "Name"

' Set up dialog event trapping:
preferred = 0 ' preferred customer flag
finished = 0 ' exit flag
ON DIALOG GOSUB DoAddress
DIALOG ON
WHILE finished = 0
WEND
DIALOG OFF
WINDOW CLOSE 2
MENU
RETURN

' Process events:
DoAddress:
myevent = DIALOG(0) ' Find out what occurred
itemhit = DIALOG(1) ' ...
myfield = DIALOG(2) ' ...
DIALOG OFF ' No interrupts to confuse things

```

Continued

Continued

```

IF itemhit = 1 THEN finished = 1 ' OK
IF itemhit = 2 THEN finished = 1 ' Cancel
IF myevent = 1 THEN ' button clicked
' radio buttons:
IF itemhit = 7 THEN
Sales = 7 ' Store for later use
BUTTON 7,2 ' Cycle radio button status
BUTTON 6,1
BUTTON 5,1
BUTTON 4,1
END IF
IF itemhit = 6 THEN
Sales = 6 ' store for later use
BUTTON 7,1
BUTTON 6,2
BUTTON 5,1
BUTTON 4,1
END IF
IF itemhit = 5 THEN
Sales = 5 ' store for later use
BUTTON 7,1
BUTTON 6,1
BUTTON 5,2
BUTTON 4,1
END IF
IF itemhit = 4 THEN
Sales = 4 ' store sales level for later use
BUTTON 7,1
BUTTON 6,1
BUTTON 5,1
BUTTON 4,2
END IF

' check box item:
IF itemhit = 3 THEN
IF preferred = 1 THEN ' unset button, flag
preferred = 0
BUTTON 3,1
ELSE
preferred = 1 ' set customer flag
BUTTON 3,2
END IF
END IF
END IF
IF myevent = 7 THEN ' TAB in edit field
theitem = myfield + 1 ' increment pointer
IF theitem > 10 THEN theitem = 1
EDIT FIELD theitem
END IF
IF myevent = 2 THEN ' mouse click in edit field
EDIT FIELD myfield
END IF
' Extract strings from editline items. To use these, you can

```

Continued

EASYDRIVE. **THE EASY HARD-DISK MANAGER.**

So, your going to buy a hard-disk. The storage capacity is great, and it's easy to use, once you've got it going. But how will you keep it organized, launch an application, or use directories and subdirectories? It can be tedious, frustrating work.

We have the answer. EasyDrive, the top selling software interface for the hard-disk. EasyDrive is **super user friendly**.

EasyDrive automatically installs your programs so that you can choose the applications you want from the EasyDrive menu. Running, removing, backing up, restoring, indexing, copying, moving files, and dozens of other functions are performed on screen. You select the commands. EasyDrive does the work.

EasyDrive comes with full documentation, ProDOS hand book, and is ProDOS 8, 16, and GS/OS compatible. We're continually updating EasyDrive with new, exciting features to keep pace with your changing needs. Watch for updates!

EasyDrive \$69.95

a product of Q Labs 313/331-0941

Available from: Quality Computers 1-800-443-6697

Also available from: Roger Coats, Silicon Express, N.A.U.G. or your favorite dealer

Continued

```
' do a simple PRINT Name$, etc. :
Name$ = EDIT$(1)
Phone$ = EDIT$(2)
Areacode$ = EDIT$(3)
Zip$ = EDIT$(4)
Nation$ = EDIT$(5)
State$ = EDIT$(6)
City$ = EDIT$(7)
Street$ = EDIT$(8)
Company$ = EDIT$(9)
Title$ = EDIT$(10)
DIALOG ON
RETURN
' END OF source code listings.
```

enough to get an idea of what you want, but not nearly good enough to come up with professional results.

To do that, you'll have to start with your best guess, make allowances for the varying pixel sizes of each letter in each field that's to be displayed, then fiddle with exact screen locations through a couple of dozen recompilations. Ugh.

Then there are layout utilities, but they're no perfect solution. Using a program such as **Dialog Layout Utility**, you can create each dialog element and drag it to the screen position you want. When you're done, you can save the layout as Pascal or assembly-language source code. Ostensibly, someone using one of these languages should be able to paste this output code directly into a program, but in practice the output isn't up to par for serious programs. The one feature it does have, however, is precise screen locations.

So in half an hour, you can lay out your dialog and get a nice printout or text file showing where everything should go. You'll still have to do the programming, but there's really nothing hard about that—write a

few lines of code and paste those lines into your source file again and again. Just change the screen location for each item, and perhaps its type and default status.

There are a number of "mouse locator" desk accessories available for download and from user-group libraries. While you may think one of these utilities will serve the same purpose as DLU, it's not true. First, they don't account for the size of an entire dialog item including text and graphics. And, just as important, they produce global screen coordinates, as opposed to the local screen coordinates you'll need to position anything in a screen window.

DLU is freeware available from most on-line services. You may contact its author, Scott Aitken, at S.AITKEN on GENie, or S.AITKEN on America Online.

The future of dialog design lies in resource management, built into the language compiler. With such a compiler, you'd design a dialog by going through a point-and-click routine, which would subsequently generate the source code for the task at hand. This is a welcome, although untried, idea that is being explored now on the Macintosh, and less successfully on the GS. It'll be a while before any standards appear, and longer yet until it means anything to BASIC.

Until that time, we have a firm understanding of one of the most elegant tools in the IIGS, and we have robust examples enabling us to fully implement dialogs of our own. Where do you go next?

ON ALERT

We've discussed only the most common dialogs—modal dialogs. You ►

Now for AppleWorks 3.0! **REPAIRWORKS.** **DON'T LOSE DATA WITHOUT IT.**

In a perfect world a program like RepairWorks wouldn't be necessary. Unfortunately the world isn't perfect and for those who have peered tearfully into a monitor filled with the dying gasps of their precious work, it can almost seem cruel. But, don't despair! RepairWorks can soften the blow of cruel fate when it involves your AppleWorks files.

RepairWorks examines your AppleWorks files and surgically removes the offending problems, reducing or eliminating the need to recreate your work.

"When my AppleWorks crashed, I was looking at hours of rebuilding time. RepairWorks turned my hours into minutes. Thank you RepairWorks."

Bruce Bauslaugh, Vero Beach, FL

"I wish RepairWorks had been around a few years ago when I was writing my very first feature for inCider."

Lafe Low, inCider Magazine

RepairWorks \$39.95

a product of Q Labs 313/331-0941

Available from: Quality Computers 1-800-443-6697

Also available from: Roger Coats, Silicon Express, N.A.U.G. or your favorite dealer

inCider Magazine
EDITORS' CHOICE
OCTOBER 1989

can explore the use of alerts, too—dialogs you can program to react at increasing levels of interactivity based on up to three repetitions of an action (often, three repetitions of a mistake). Note that alerts require only minimal logic to check for user input, and only one informational field more than the lowest-level dialog—a field specifying which alert graphic to use.

Modeless dialogs, which act as a standard window available at all times, are another challenge. The actual logic for a modeless dialog is covered here. The added concern in this case is monitoring all windows that might be available on the desktop at any one time. You'll have to have in your main program logic a way of knowing when the dialog window has been selected. (You can do this most easily with an empty global variable set when a mouse-down event occurs in the dialog window.)

PRODUCT INFORMATION

AC/BASIC

Absoft Corp.
2781 Bond St.
Rochester Hills, MI 48307
(313) 853-0050
\$125

Micol Advanced BASIC

Micol Systems
9 Lynch Road
Willowdale, Ontario M2J 2V6
Canada
(416) 495-6864
\$145

TML BASIC

TML Systems
8837-B Goodbys
Executive Drive
Jacksonville, FL 32217
(904) 636-8592
\$125

DETAIL WORK

Ultimately, you can create full custom dialogs to meet needs only you can imagine. To do this, you'll need to fully understand your compiler's memory-management techniques (AC/BASIC won't do in this case), as well as QuickDraw II screen drawing, and perhaps even graphics design and animation algorithms.

You can take this as far as you want. The one thing that's most important to remember, no matter what your goal, is to execute the details right. Your goal is within your grasp. □

CONTRIBUTING EDITOR JOE ABERNATHY IS A JOURNALIST WITH *THE HOUSTON CHRONICLE*. HE'S A CERTIFIED APPLE DEVELOPER AND THE AUTHOR OR COAUTHOR OF EIGHT APPLE II PROGRAMS. WRITE TO HIM C/O *INCIDER*, 80 ELM STREET, PETERBOROUGH, NH 03458. ENCLOSE A SELF-ADDRESSED, STAMPED ENVELOPE IF YOU'D LIKE A PERSONAL REPLY.

Now for AppleWorks 3.0! **SUPERPATCH.** **DO-IT-YOURSELF APPLEWORKS.**

Q Labs announces SuperPatch 6.1, the world's most comprehensive customization program for AppleWorks 2.0, 2.1 and now, for AppleWorks 3.0!

**SuperPatch 6.1 installs over 100 patches on AppleWorks 3.0,
and over 150 patches on AppleWorks 2.0 and 2.1.**

SuperPatch is menu driven, and simple to use. SuperPatch will also de-install alterations, enabling you to try various patches for as long as you like, and easily remove some or all of them later. Plus, SuperPatch modifications are compatible with your AppleWorks modifications.

Here are a few of the patches available with SuperPatch.

Time/Date Display
Automatic Time/Date in Reports
No Space Bar on Boot Up
Error Tone Customization

Printer Modifications
Change Cursor to any Mouse Character
Cursor Blinker Rate Modification
And Many, Many More!

SuperPatch \$39.95

a product of Q Labs 313/331-0941

Available from: Quality Computers 1-800-443-6697

Also available from: Roger Coats, Silicon Express, N.A.U.G. or your favorite dealer