The Electronic Magazine for Macintosh™ Developers

Issue #2

6/7/85

Contents

Advertiser Index:

This issue uses the following fonts: New York 9, 10, 12, and 18 point; and Geneva 9 and 10 point. If you are going to print the magazine, these fonts and the sizes twice as large should be present.

MacDeveloper is in no way sponsored by or associated with Apple Computer, Inc.

MacDeveloper is published the first Friday of each month. Distribution is via electronic bulletin board systems and national information services. If these avenues of distribution are not accessible to you, send a self-addressed, stamped envelope with a Macintosh diskette to the following address; unless you request a different issue, the latest issue of MacDeveloper will be returned.

Harry Chesley
1850 Union Street, #360
San Francisco, CA 94123

Special notice to BBS operators: You are actively encouraged to post this magazine on your BBS, so long as all of it is posted. The larger the distribution of MacDeveloper, the more articles and advertising we will get to support the magazine, to everyone's benefit.

MacDeveloper

The Hacker Ethic: Telling Write from Wrong

An Editorial

There's been a lot of talk lately about the **Hacker Ethic**. The magazines are all writing about it. There are several books out. There was even a Hacker Convention last year.

Part of the Hacker Ethic is supposed to be: "Share Information." (I think that's second only to: "Take Everything Apart.") When I started this magazine, I therefore thought I would have no trouble getting articles for it. The Mac is a hacker's machine (don't tell "the rest of them"). Hackers love to share information. They'll all rush to write articles for MacDeveloper. Write?? Wrong!!

Instead I'm desperate for articles. All of these so-called hackers are apparently hoarding their Macintosh secrets, keeping those hidden RAM locations stashed for their own private use, jealously guarding the prime algorithms so no one else can beat them to the juicy markets. How can these people hold their heads up at local user's group meetings?

OK. By now the real hackers have cried out in shame, stopped reading, and are off writing the articles you'll be reading next month. So I can stop flaming and tell you what I really think about the Hacker Ethic.

I too fit the cultural stereotype of the hacker. I too spent all of high school and college in rooms with no windows and squinted painfully whenever exposed to sunlight. I too hacked the great systems of the past — in my case, it was the ARPANet rather than the phone system — though my definition of hacking is less destructive than some people's. I too switched over to personal computers, and saw them as the answer to decades of prayers — anyone want to buy a used Apple I?

But I don't like being shoved into a stereotype, no matter how snug and cozy the fit. Stereotypes are simply excuses for not listening to what a person's really saying. So, I don't really believe any of that stuff at the start of this editorial about sharing information. Hackers are just people. Some share. Some hoard. Some even vandalize.

I'm in the share camp. I'd rather loose a few ideas than live my life worrying about loosing them. And one of the things I've learned over the fifteen years I've been hacking is that there'll always be another good idea.

So those of you who're in the share camp with me, how about writing some good articles about how to develop software for the Macintosh.

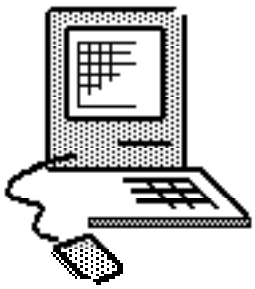
--Harry Chesley

Solutions,Inc.

~Presents~

Macintosh Design Workshop

Design of User Interface and Internal Structure —
A Three Day Seminar for Macintosh
Designers and Programmers



Chicago ... July 17-19
San Jose ... September 16-19
London ... October 7-9

Call or write Solutions now to register or for more information at (802) 229-0368, Box 989 Montpelier, Vermont 05602, or MCI: Solutionsinc

News, Notes, and Letters

News

It is not the intention of this magazine to report news in the sense of time-value information. We are more interested in reporting information of longer-term value, such as how to program the Macintosh. However, from time to time, news comes along which just can't be ignored...

This month it was at the Stanford Macintosh User Group's MacFest '85. In a session which brought together many of the people in the original Macintosh team, Bud Tribble, currently Macintosh Software Manager, said that the main change in the new ROMs would be a true hierarchical file system. There will be some bug fixes, etc., as well, but this is the one really substantial change. It was likened to the MS-DOS change from version 1 to version 2, in which Microsoft made a similar change. He did not, however, say when the new ROMs would be released.

This is good and bad news. The good part is that the Macintosh badly needs such a hierarchical file system, especially for use with hard disks. The bad part is that Macintosh software developers will now have to deal with two different versions of the file system. How bad it is we'll have to wait to see, but it's almost certainly worth the price.

Notes and Letters

Warning!

There is a company called LisaVision out here in California that is selling a product by the same name. It is supposed to correct the pixel distortion on the Lisa running MacWorks, and be able to switch to native Lisa video mode for running 7/7 or the Workshop. Well, it's not worth it! It retails for $79.95, and consists of about $6.75 worth of parts. You can fry yourself to a crisp installing it if you don't read the directions properly, or worse, you can damage your Lisa! The instructions should have tipped me off. The first thing you're supposed to do is "...find the horizontal width and vertical height adjustments on the Lisa's video board. Adjust hor. fully counterclockwise, and vert fully clockwise." Well, that is the meat and potatoes of the whole $79.95 package. Their "hardware" consists solely of some jumpers and a switch. One of the adjustments on their switch does nothing, and the other barely made a difference. The jumpers aren't shielded, and are attached by alligator clips. After installing the switch, my Lisa's display was experiencing considerable waviness.

The long and short of all this is do not purchase this device. To their credit, they do not say they offer a "...100% solution." However, they do not volunteer that fact

unless you ask them. Modern day consumers rarely ask if the tires are included in the price of a car, and I guess LisaVision is capitalizing on that fact!

Anyway, "caveat emptor" (buyer beware!). BTW, when did Apple say they would have a "solution"...

Steve (a graduate from David Horowitz's school of consumerism) Maller

---

Date: 18-May-85 19:18
From: Brian Jay Wu [75216,3556]
Subj: Aztec C Review

Read your first issue of macintosh development. Not bad. I have several comments on your Aztec review.

Firstly, a minor correction - the compiler defaults to producing the expected object code, so an explicit assembler pass is not necessary. However, I have noticed that for "production" compiles, it is best to generate an explicit assembler file and then use the "-s" option during assembly, which seems to optimize to a degree.

Next, the Aztec stdio.h file unfortunately insists on using their routines agetc and aputc, which do horrible canonicalization to the input. Specifically, the agetc/aputc anomalies remove such things as carriage returns, so if your code EXPECTS carriage returns, forget it!

As for the code provided by Manx, most of the C code is not very pleasing in an stylistic sense - Jim Goodnow II (who I believe writes their stuff) doesn't seem to believe in comments...

As far as Manx's customer relations go, they are not necessarily as excellent as your experience would seem to indicate. For instance, I waited almost three months before getting Release Delta, and then got deluged in no fewer than TWO copies of it! At least, their product is so superior I can forgive them...

Lastly, may I announce that I am now shipping as public domain software a set of "tools" for use under Aztec. Some highlights - an lpr utility, and an lsarcv (do an ls on .arc files...). Executables can be had in dl3 under MAUG, but the full release includes source code too!

Brian Jay Wu

CIS 75216.3556

PO Box 203
Newbury Park, CA 91320

Date: 31-May-85 21:07
From: Chris Allen - DOTP [76703,472]
Subj: Publish Your Software!!

*** Freelance Programmers Wanted! ***

If you are a freelance programmer, or feel that you want to make some money from your computer hobby, we at Dreams of the Phoenix, Inc. would like to invite you to submit your utilities, desk accessories, and applications for publication.

Currently, Dreams of the Phoenix, Inc. is publishing programs from over a dozen freelance programmers. So far, four products have been released: Day Keeper Calendar, an appointment scheduler and time accounting application; Mouse Exchange BBS , a remote bulletin board system; Mouse Exchange Terminal, a multi-terminal emulation application; and Quick & Dirty Utilities Volume One, a grab bag of desk accessories and utilities, including a disk cataloger, font manager, xmodem terminal desk accessories, a super-note-pad desk accessory, and much more. All of our products have been very favorably reviewed!

Future products include: another volume of Quick & Dirty Utilities, List Keeper Database, Note Keeper Organizer, Calc Keeper Spreadsheet, and a games disk!

The copyright in your software will belong to you - Dreams of the Phoenix purchases only the marketing rights. Royalties range from 1 to 2% of net receipts for small utilities, to 15% for complete and tested applications with documentation. The typical royalty percentage is 10%, as we work closely with you during the design, and in the polishing, testing, and
documenting of your software.

All Dreams of the Phoenix products are stand-alone applications (no BASIC or Pascal should be needed), are not copy-protected, and will retail for $39.95. We have high standards for our products, thus we have worked out a "Guide for Freelance Programmers" to aid you in your development of software.

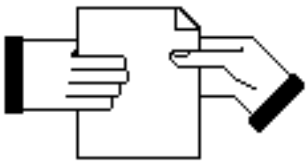
You can find a copy of our "Guide for Freelance Programmers" in the Mac Developer's Forum (PCS-7) Data Library 7. Or you can send a EMAIL to:

Chris Allen 76703,472

or write:

Dreams of the Phoenix, Inc.
Att: Christopher Allen
P.O. Box 10273
Jacksonville, FL 32247
(904) 396-6952

MacDeveloper



MacInTouch

The Newsletter for Apple® Macintosh™ Professionals

For

developers, dealers, corporate use rs, Mac Office workers, hackers.

Technical Info	Fresh
Market Research	Unbiased
Network Summaries	Concise
News	Intelligent
Tips & Bug Listings	Clear
References	

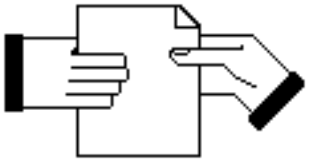
Ford-LePage, Inc.
PO Box 786
Framingham, MA 01701

617/527-5808
Delphi "MACINTOUCH"
CompuServe [75056,1225]

Mailed monthly:

U.S.: $48/year
Canada: $70
Europe: $85

(US Subscriptions prior to Sept. are just $40. Quantity discounts are also available.)



What Every Application Should Know

Harry R. Chesley

Introduction

Constructing a Macintosh application involves a great number of concepts used in an even larger number of calls to Toolbox routines. The application programmer must understand and manipulate events, windows, menus, keystrokes, and keyboard commands; then he can get on with the job of programming the specific application.

Some of these Toolbox calls are generic to all applications. That is, there is a framework of Toolbox interactions which reappears in virtually every application, with only minor variations. This article builds and explains this generic framework.

The same information can (more or less) be gleaned from the examples provided by Apple in Inside Macintosh. This article, however, collects all of the information in one place, and provides a more complete explanation of why the various actions are taken.

It is assumed that the reader has at least skimmed the salient portions of Inside Macintosh. No attempt is made to explain all of the intricacies of the Toolbox, or even of the routines called in the code fragments presented. As always, Inside Macintosh should be consulted for more details.

All of the code examples are given in Pascal to make it easier for the reader to use this article in conjunction with Inside Macintosh, which is all written assuming Pascal. They have not been tried verbatim, but the equivalent code in C has been tried in (almost) all cases.

Initialization

Before use, many Toolbox managers need to be initialized. Not every initialization routine needs to be called for every application, but it is simpler and easier to take this "shotgun" approach.

The following piece of code initializes a Macintosh application:

```
InitGraf(@thePort);          Initialize Quickdraw.
InitFonts;                   Initialize the font manager.
InitWindows;                 Initialize the window manager.
FlushEvents(everyEvent,0);   Clear any pending events.
TEInit;                      Initialize TextEdit.
InitDialogs(NIL);            Initialize the dialog manager.
InitCursor;                  Initialize the cursor.
initAppl;                    Do any application-specific initialization.
initMyMenus;                 Initialize the menus.
```

Most of the routines above simply initialize their respective Toolbox areas. Two need a little more explanation, and two are application supplied.

InitGraf must be passed the address of a block of memory where it can put its global variables. Generally, the compiler Toolbox support provides for allocating such an area and naming the component variables, the first of which is thePort. Whether you use the variables in this area or not, you or the compiler must provide the space for it.

FlushEvents clears out any events still pending from previous activity on the Macintosh. For instance, an impatient user who types ahead while the program is still loading. This isn't essential, but is generally a good idea.

The initAppl routine performs any application-specific initialization.

initMyMenu initializes the menus. It is done last so that the menu-bar titles only appear when the application is ready for use. This is less confusing for the user than having them appear but not be useable.

initMyMenu builds the menu list from menu definitions in the resource file and from the available desk accessories, and then displays them. For the purposes of this article, we will use only the three standard menus: the Apple menu, the "File" menu, and the "Edit" menu.

First, the MENU resources that describe these menus must exist in the program's resource file. They have the following format, assuming that RMaker is used to construct the resource file:

```
TYPE MENU              The Apple menu.
,256
\14                    The Apple symbol.
About this program...
( -                    A dividing line.

TYPE MENU              The File menu.
,257
File
New
Open...
```

```
Close
Save
Save as...
( -
Print...
Page Setup...
( -
Quit

TYPE MENU              The Edit menu.
,258
Edit
Undo
( -
Cut
Copy
Paste
Clear
```

The menu definitions must now be used to construct the menus in memory. The Apple menu is a special case, as it must use AddResMenu to add in the desk accessories:

```
VAR
      myMenu: array [1..3] of MenuHandle;    A place to store the menu
handles.

InitMenus;                                   Init the Toolbox menu manager.

myMenu[1] := GetMenu(256);                   Get the Apple menu.
AddResMenu(myMenu[1],'DRVR');                Add in the desk accessories.

myMenu[2] := GetMenu(257);                   Get the File menu.

myMenu[3] := GetMenu(258);                   Get the Edit menu.
```

(**Note**: Since they are so small, the code fragments in this article are using numbers rather than defined constants. Don't be fooled into thinking this is good programming practice.)

Now that the individual menus have been constructed, we can link them into the menu list. This is done by repeatedly calling InsertMenu to insert each menu at the end of the list. Finally, we call DrawMenuBar to actually display the menus.

```
for i := 1 to 3 do InsertMenu(myMenu[i],0);    Insert the menus.

DrawMenuBar;                                   Draw the menus.
```

Finder File Name Parameters

In some cases, the Finder may pass a file to be processed to the application. This happens when either a document is opened or printed, or when a document and application are selected together and opened. The application finds out about these files by calling CountAppFiles, which returns the number of files (there may be several) passed; it can then call GetAppFiles to get the individual file names. The following code determines if files were passed, and calls eventLoop if none were (i.e., starts the normal command processing) or doFiles if there were files.

```
VAR  openOrPrint, numberOfFiles: integer;

CountAppFiles(openOrPrint,numberOfFiles);     Get the file count.

if numberOfFiles > 0 then doFiles(openOrPrint,numberOfFiles)
else eventLoop;
```

(The openOrPrint parameter tells whether the Finder requested that the application print the files or just open them.)

In most applications, when the request is to print the file, not all of the initialization described in the previous section need be done. In particular, building and displaying the menus can be skipped. For some applications, this is true even when the files are to be opened. Exactly how this works depends on the type of application and is beyond the scope of this document.

The Main Event Loop

Once everything has been initialized, and the application has decided that it should respond to user input (rather than just performing some function like printing a file and exiting), it enters the main event loop.

The event loop receives and decodes events, which report a wide range of external changes, from keyboard or mouse entries to window update requests.

Also in the event loop are calls to any procedures which must be called periodically, such as the system procedure SystemTask, an application procedure for any application-specific periodic tasks, periAppl, and an application procedure to change the cursor shape as it enters different parts of the screen. A discussion of TextEdit is beyond the scope of this article, but it should be noted that if the application needs to call TEIdle, it should be done within periAppl.

The main event loop, therefore, looks like this:

```
VAR
        done, gotEvent: boolean;
        newEvent: EventRecord;

done := FALSE;

repeat                          Loop until done.

        SystemTask;             Let the system do its stuff.

        periAppl;               Application-specific periodic tasks.

        updateCursor;           Change the cursor shape if appropriate.

        gotEvent := GetNextEvent(everyEvent,newEvent);
        if gotEvent then
                case newEvent.what of
                        ...Handle event...
                end;

until done;
```

Handling the event is, of course, the heart of the event loop. It consists of a series of cases for each type of event that might occur. Each case will be considered separately below:

Mouse Events

Applications generally ignore mouse up events (user letting go of the mouse button), but mouse down events (user pressing the button) are the most complicated events, at least as far as generic handling goes.

First, FindWindow is called to find out which window the mouse was pointing at at the time the user pressed the mouse button. FindWindow returns not only the window, but also which part of the window. This part code is used to further classify the event.

For mouse presses in a system window, FindWindow returns inSysWindow. This type of event is handled by the system, so the application just calls SystemClick. (Note: A "system window" is really just a desk accessory.)

For mouse presses in the menu bar, FindWindow returns inMenuBar. The application should then call MenuSelect, which tracks the mouse, pulls down the appropriate menus, and highlights the selected menu entries. If the mouse is released in an enabled menu item, an application routine is called to perform the requested operation.

For mouse presses in the drag bar of a window, FindWindow returns inDrag. Generally, this type of event can be handled generically, by calling DragWindow with

a standard rectangle in which any window can be dragged. Some applications, however, may need to limit the drag rectangle of some windows more than of others. Before calling DragWindow, the application must check if this window is in front and select it if not. Also, the drag rectangle must be defined: this can be done using screenBits.bounds, which contains the total screen size of the Macintosh running the program; this makes the application more portable across the current and future Macintosh models.

If the mouse button is pressed in the interior of the window, FindWindow returns either inContent or inGrow (if it's in the grow box at the bottom right corner). In either event, if this window is not the front-most window, it should be made so. If it is, an application routine is called to handle the event. The grow window case could almost be made generic, but it often involves moving and/or resizing controls.

Finally, if the mouse if pressed in the "go away box" of the window, FindWindow returns inGoAway. TrackGoAway is called first to determine if the mouse is still pointing at the go away box when released. If it is, then an application routine is called to close the window and clean up any related data structures.

Putting all of this together, the following piece of code handles the mouse down events:

```
VAR
        whichWindow:  WindowPtr;
        FWReturnCode: integer;
        dragRect: Rect;

mouseDown:              From newEvent.what case statement.

        FWReturnCode := FindWindow(newEvent.where,whichWindow);

        case FWReturnCode of

                inSysWindow:  SystemClick(newEvent,whichWindow);

                inMenuBar: doCommand(MenuSelect(newEvent.where));

                inDrag:
                        begin
                        if whichWindow <> FrontWindow then
SelectWindow(whichWindow);
                        dragRect.top := 24; dragRect.left := 4;
                        dragRect.bottom := screenBits.bounds.bottom-4;
                        dragRect.right := screenBits.bounds.right-4;
                        DragWindow(whichWindow,newEvent.where,dragRect);
                        end;
```

```
                inGrow:
                inContent:
                        begin
                        if whichWindow <> FrontWindow then
SelectWindow(whichWindow)
                        else
                                if FWReturnCode = inContent then
                                        applMouseDown(newEvent,whichWindow)
                                else  applGrowWindow(newEvent,whichWindow);
                        end;

                inGoAway:
                        if TrackGoAway(whichWindow,newEvent.where) then
                                applGoAway(whichWindow);

        end;
```

Keyboard Events

There are three types of keyboard events: keyDown, keyUp, and autoKey. As with mouse up events, key up events are generally ignored by an application. The distinction between keyDown and autoKey events is that autoKey means that the keyboard input was generated by the auto-repeat feature of the keyboard. It is good practice to ignore auto-repeat keys which would be interpreted as keyboard commands (menu equivalents).

To handle keyboard events, then, we first check whether the command key was held down (a menu equivalent keyboard command). If so, MenuKey will translate the keyboard input into the equivalent menu selection, which can then be passed to the previously referenced application procedure doCommand. If not, or if it was not a valid command key, an application procedure is called to handle the keyboard event.

```
VAR
        key: char;
        keyCommand: longint;

case keyDown, autoKey:        From newEvent.what case statement.

        begin key = myEvent.message mod 256;
        if BitAnd(newEvent.modifiers,CmdKey) <> 0 then keyCommand =
MenuKey(key);
        if newEvent.what = keyDown and BitAnd(newEvent.modifiers,CmdKey)
<>0
                and HiWord(keyCommand) <> 0 then doCommand(keyCommand);
        else applKeyDown(newEvent);
        end;
```

Activate, Deactivate, and Update Events

These events have to do with the window manager. When a new window is selected, the old one is deactivated and an event generated to notify the program; the new window is

then activated and another event generated for that action. When a window needs updating, because it became active, was moved, or another window was moved, exposing parts of the window not previously exposed, an update event is generated.

Note: Window activation and updating is an extremely important part of the Macintosh user interface, but little of it is generic. Macintosh programmers should study this part of the Macintosh toolbox carefully to be sure that their application is implemented properly in this area.

All of these window-related events are application-specific, but the decoding of the events occurs as follows:

```
case activateEvt:          From newEvent.what case statement.

     if BitAnd(myEvent.modifiers,1) <> 0 then applActivate(newEvent)
     else applDeactivate(newEvent);

case updateEvt:

     applUpdate(newEvent);
```

Disk Events

Disk events are generated when the user inserts a disk into the drive. The system automatically attempts to mount the disk. If the mount succeeds, nothing more needs to be done. If the mount fails, however, the application must call DIBadMount to ask the user if he wants the disk initialized, and to do the initialization if so. The high word of the event message contains the return code from PBMountVol, which is equal to noErr if the mount went OK. The application must also tell DIBadMount where to put the initialization dialog box.

```
case diskEvt:          From newEvent.what case statement.

     if HiWord(newEvent.message) <> noErr then
          begin DIPt.h := 100; DIPt.v  := 100;       The dialog box top-right
corner.
          DIBadMount(DIPt,newEvent.message);
          end;
```

Other Events

There are several other possible events, including network, driver, and application-defined ones. However, most applications can ignore these other events and concentrate solely on the ones described above.

Commands

Commands, whether entered via the pull-down menus or by equivalent keyboard commands, are interpreted in the doCommand procedure shown below. The long

integer passed to doCommand is generated by either MenuSelect or MenuKey. The top word contains the ID of the menu, or zero if nothing is to be done; the bottom word contains the item within the menu. Decoding the command is therefore simply a matter of a set of nested case statements.

Once decoded, most of the remaining work is application-specific. One exception to this rule is the set of desk accessory menu items. When these are selected, OpenDeskAcc needs to be called to open the accessory.

The other exception is the Edit menu: for the undo, cut, copy, paste, and clear commands, SystemEdit should be called to let desk accessories get access to these commands; SystemEdit will return TRUE if it accepted the command, and the application should therefore do nothing more with it. If the menu entries are in the order given in the resource file, SystemEdit can simply be passed the menu item number minus one.

Finally, after the command has been executed, HiliteMenu is called to turn off the menu highlighting.

```
procedure doCommand(theCommand: longint)

var
      theMenu, theItem: integer;
      name: str255;
      refnum: integer;

begin theMenu = HiWord(theCommand); theItem = LoWord(theCommand);

case theMenu of

      256:        The Apple menu.
            if theItem = 1 then applAbout
            else
                  begin GetItem(myMenu[1],theItem,name);
                  refnum := OpenDeskAcc(name);
                  end;

      257:        The File menu.
            applFileMenu(theItem);

      258:        The Edit menu.
            if not SystemEdit(theItem-1) then applEdit(theItem);

end;

HiliteMenu(0);    Turn off the menu hiliting.

end;
```

Summary

The generic structure of a Macintosh application has been presented. This has included:

- Initialization.
- Event decoding.
- Mouse-related events.
- Keyboard-related events.
- Window activation and updating.
- Command decoding.

It is recommended that the reader now look at the example application programs supplied in Inside Macintosh in order to see how this structure fits into an actual application.

Hopefully, this description of the common structure of applications in the Macintosh will help new programmers get started, and perhaps help to crystalize the knowledge of more experienced programmers.

Appendix: The Whole Program

The following is the entire generic program in one place. This appendix can be copied out and pasted into another file as the framework for a new application.

Program:

```
const

MENUNUM = 3;          (* Number of menus. *)

APPLEMENU = 256;          (* Apple menu ID. *)
FILEMENU = 257;       (* File menu ID. *)
EDITMENU = 258;       (* Edit menu ID. *)

var

done: boolean;    (* All done flag: TRUE if program should exit. *)
myMenu: array [1..MENUNUM] of MenuHandle; (* Place to store the menu
handles. *)

procedure initEverything;

begin InitGraf(@thePort);
InitFonts;
InitWindows;
FlushEvents(everyEvent,0);
TEInit;
InitDialogs(NIL);
```

```
InitCursor;
initAppl;
initMyMenus;

end;

procedure checkFinderInput;

var
    openOrPrint, numberOfFiles: integer;

begin CountAppFiles(openOrPrint,numberOfFiles); (*Get the file count. *)

if numberOfFiles > 0 then doFiles(openOrPrint,numberOfFiles)
else eventLoop;

end;

procedure initMyMenus;

var
    i: integer;

begin InitMenus;                          (* Init the Toolbox. *)

myMenu[1] := GetMenu(APPLEMENU);    (* Get the Apple menu. *)
AddResMenu(myMenu[1],'DRVR');       (* Add in the desk accessories. *)

myMenu[2] := GetMenu(FILEMENU);     (* Get the File menu.*)
myMenu[3] := GetMenu(EDITMENU);     (* Get the Edit menu. *)

for i := 1 to MENUNUM do InsertMenu(myMenu[i],0); (* Insert the menus.
* )

DrawMenuBar;      (* Draw the menus. *)

end;

procedure eventLoop;

var
    getEvent: boolean;                    (* GetNextEvent return. *)
    newEvent: EventRecord;                (* Event from GetNextEvent. *)
    whichWindow: WindowPtr;               (* Which window from
FindWindow. *)
    FWReturnCode: integer;                (* FindWindow return code. *)
    key: char;                            (* Keyboard character. *)
    keyCommand: longint;                  (* Keybd input translated to menu
equivs. *)
    dragRect: Rect;                       (* Limit for dragging windows
around. *)
```

```
    DIPt: Point;                        (* Where to put the disk init
dialog. *)

begin done := FALSE;

repeat      (* Loop until done. *)

    SystemTask;             (* Let the system do its stuff. *)
    periAppl;               (* Periodic application activity. *)
    updateCursor;           (* Change the cursor shape if appropriate. *)

    gotEvent := GetNextEvent(everyEvent,newEvent);
    if gotEvent then
        begin case newEvent.what of

            mouseDown:
                (* Get the window/window part. *)
                FWReturnCode :=
FindWindow(newEvent.where,whichWindow);
                case FWReturnCode of
                    (* If in a system window, let the system handle it. *)
                    inSysWindow: SystemClick(newEvent,whichWindow);
                    (* If in the menu bar, track it, then let doCommand handle
it. *)
                    inMenuBar: doCommand(MenuSelect(newEvent.where));
                    (* If in the drag bar, let him drag it around. *)
                    inDrag:
                        begin
                        if whichWindow <> FrontWindow then
                            SelectWindow(whichWindow);
                        dragRect.top := 24; dragRect.left := 4;
                        dragRect.bottom := screenBits.bounds.bottom-4;
                        dragRect.right := screenBits.bounds.right-4;

DragWindow(whichWindow,newEvent.where,dragRect);
                        end;
                    (* If in the interior... *)
                    inGrow, inContent:
                        (* If this window isn't in front, make it be so. *)
                        begin if whichWindow <> FrontWindow then
                            SelectWindow(whichWindow)
                        (* Otherwise, let the application handle it. *)
                        else
                            if FWReturnCode = inContent then
                                applMouseDown(newEvent,whichWindow)
                            else applGrowWindow(newEvent,whichWindow);
                        end;
                    (* If in the go away box, track and call the appl. go away
routine. *)
                    inGoAway:
                        if TrackGoAway(whichWindow,newEvent.where) then
```

```
                                applGoAway(whichWindow);
                    end;

                (* Keyboard events: *)
                case keyDown, autoKey:
                    begin key = myEvent.message mod 256;
                    if BitAnd(newEvent.modifiers,CmdKey) <> 0 then
                        keyCommand = MenuKey(key);
                    if newEvent.what = keyDown and
                        BitAnd(newEvent.modifiers,CmdKey) <> 0 and
                        HiWord(keyCommand) <> 0 then
doCommand(keyCommand);
                    else applKeyDown(newEvent);
                    end;

                (* Activate/deactivate window: *)
                case activateEvt:
                    if BitAnd(myEvent.modifiers,1) <> 0 then
applActivate(newEvent)
                    else applDeactivate(newEvent);

                (* Update window: *)
                case updateEvt:
                    applUpdate(newEvent);

                (* Disk insertion event: *)
                case diskEvt:
                    if HiWord(newEvent.message) <> noErr then
                        begin DIPt.h := 100; DIPt.v  := 100;
                        DIBadMount(DIPt,newEvent.message);
                        end;

            end;
        end;

until done;  (* End of main event loop. *)

end;

procedure doCommand(theCommand: longint)

var
    theMenu, theItem: integer;
    name: str255;
    refnum: integer;

begin theMenu = HiWord(theCommand); theItem = LoWord(theCommand);

case theMenu of

    APPLEMENU:          (* The Apple menu. *)
```

```
            (* Check for "About this program..." *)
            if theItem = 1 then applAbout
            else
                (* Otherwise find and open the desk accessory. *)
                begin GetItem(myMenu[1],theItem,name);
                refnum := OpenDeskAcc(name);
                end;

        FILEMENU:           (* The File menu. *)
            applFileMenu(theItem);

        EDITMENU:           (* The  Edit menu. *)
            (* Check for accessory edit; if not, call the application edit. *)
            if not SystemEdit(theItem-1) then applEdit(theItem);

    end;

    HiliteMenu(0);      (* Turn off the menu hiliting. *)

    end;
```

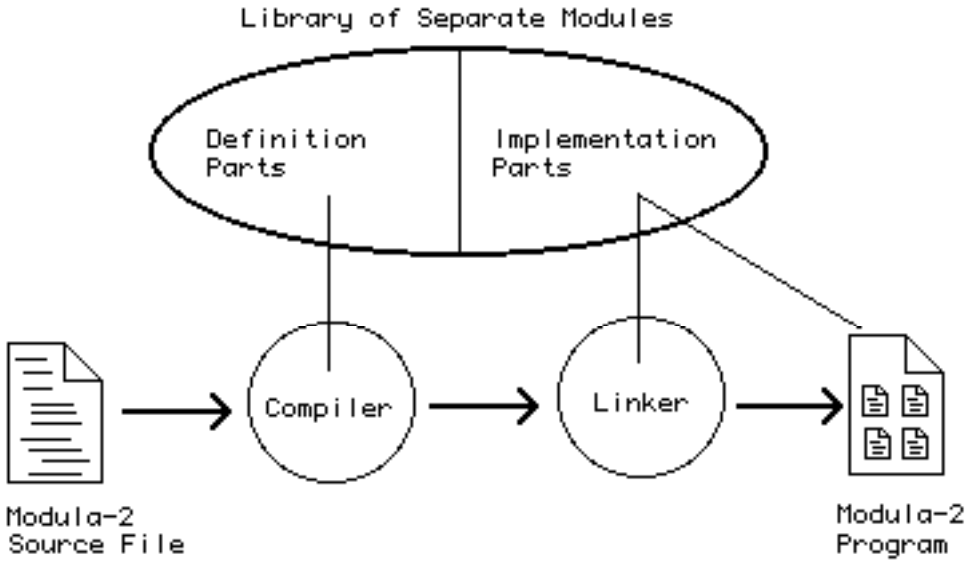
Resources:

```
* The Apple menu:
TYPE MENU
,256
\14
About this program...
( -

* The File menu:
TYPE MENU
,257
File
New
Open...
Close
Save
Save as...
( -
Print...
Page Setup...
( -
Quit

* The Edit menu:
TYPE MENU
,258
Edit
Undo
```

(-
Cut
Copy
Paste
Clear

MacModula-2™

Modula-2 Development System for the Macintosh®



$150.00

- Full Modula-2 Implemetation
- Mac-like System: Integrated w/ Finder
- Modula-2 Programs use Menus, Windows, etc.
- Supports over 400 QuickDraw and ToolBox ROM Routines
- Editor, Compiler, Linker, & Library
- Extensive Documentation of Modula-2 System
- Works on 128K or 512K Mac (2 drives recommended)
- Disks not copy protected
- No developer license fees or royalties

Modula Corporation
950 North University Avenue
Provo, UT 84604
(801) 375-7400
Toll Free Outside Utah (800) LILITH2

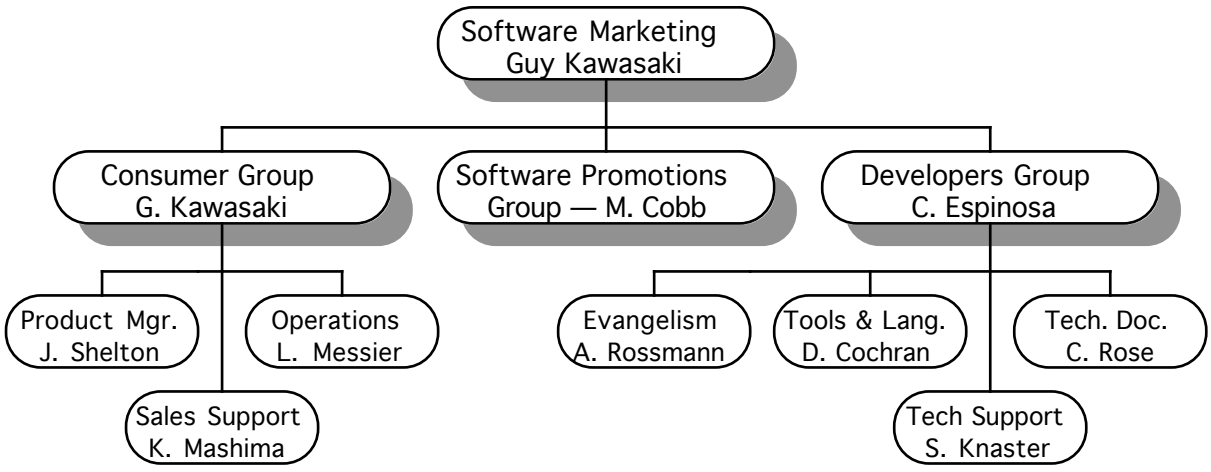
Outside Outside Macintosh

The following articles are reprinted from **Outside Macintosh**, Apple's Newsletter for Certified Developers, with permission from Apple.

◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

Software Marketing Defined

By Guy Kawasaki, Software Marketing Manager



In this issue of *Outside Macintosh*, I would like to explain the Software Marketing organization and to introduce Chris Espinosa, the new manager of the Macintosh Developers Group. Software Marketing is currently divided into three areas: Consumer Group, Software Promotions Group, and Developers Group.

Consumer Group

The Consumer Marketing Group has product management responsibilities for Macintosh consumer products. Joe Shelton heads up a staff of product managers for Apple software, such as MacWrite™, MacPaint™, and MacProject™. Kyle Mashima

and his Sales Support Group ensures maximum exposure and acceptance for Apple and third-party software with the Apple sales force and dealers through training and demo materials. And Linda Messier's Operations Group provides the logistical and operational support to make it all happen.

Software Promotions Group

Matt Cobb's name is familiar to all of you. His Promotions Group plans and executes marketing programs for both Apple and third-party products. This group was responsible for the "Software Sampler" and "Own-a-Mac Coupon Program" last fall, as well as the recent *Wall Street Journal* event. Patti Barrus, one of the team members, orchestrates the co-marketing packets and Macintosh developer mailings. This Group would like to hear from you several months before your product is introduced.

Developers Group

I would like to introduce Chris Espinosa, our new Developer Group Manager. As Apple employee #8, Chris brings a wealth of experience to the group. He got hooked on computers when he was in junior high school, and started with Apple while he was still in high school (ask him sometime how he got his job at Apple — it's a story in itself). Back then, Chris started out writing demos and answering the tech support hot line. By the time he was ready for college, he was also ready to write his first technical reference manual — and he started a tradition. The *Apple II Technical Reference Manual* set the standard for all future apple manuals, considered by many to be the best in the industry.

Chris went on to write more manuals. He later took charge of the Macintosh User Education Group, the people who write all the Macintosh manuals — and more important to you, *Inside Macintosh*. Chris is bringing the technical documentation group with him, and he will manage the group known to you as the Macintosh Developers Group. Welcome aboard, Chris!

Here is just a brief description of the Macintosh Developers Group, comprised of four functions: Evangelism, Technical Support, Development Tools and Languages, and Technical Publications.

Evangelism

Alain Rossmann, manager of the software evangelists, has a clear mission — to make sure there is an abundance of quality software available for the Macintosh. You should contact his group if you have any questions about new-product development, Macintosh user interface, or if you would like guidance on potential AppleTalk and LaserWriter products. Once you become certified, the software evangelists are your entry point into the Macintosh Developers Group.

Development Languages and Tools

Dan Cochran's group is responsible for products related to Macintosh development environments, and works closely with all of the system software developers. Dan's vision is to provide information and tools electronically to all developers. He is currently working with a number of on-line services to achieve that goal.

Technical Support

Scott Knaster's group has the mammoth task of responding to the hundreds of phone

calls and letters from registered developers seeking technical support. His group provides *Tech Notes* and answers MCI-Mail for developers who are certified but not registered. It's also famous for sponsoring MacCollege — three days of intense labor for Macintosh developers.

Technical Documentation

Caroline Rose heads up the Technical Documentation group, which is responsible for *Inside Macintosh* and other technical documentation.

From all of us, we look forward to a long and mutually beneficial relationship — please keep in touch.

◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

Languages and Development Tools Update

By Dan Cochran, Languages and Development Tools Manager

The entire month of March and the better part of April found your humble reporter on the proverbial road. A very successful series of developer seminars in Palo Alto, Chicago, and Boston; a rather nondescript trade show in Atlanta (Softcon); and miscellaneous business in less-well-known geographic locations have reset my biological time to that of somewhere between Upper Volta and New Delhi.

Naturally, the first task awaiting me on my Cupertino homecoming was an article (deadline fast approaching) for this distinguished publication. Therefore, I'll do my best to fill you in on what's new, dispel some old rumors, and possibly start some new ones.

MDS Ships

Assembly-language diehards, salvation is at hand. The first Apple-labeled native Macintosh development environment is now available through both retail channels and the Certified Developer Program. The Macintosh 68000 Development System (MDS) is a complete assembly-language development environment that lets you write programs for the Macintosh on the Macintosh. MDS includes a window-based editor, macroassembler, linker, resource compiler, powerful set of debuggers, useful utilities, and all the traps and equates you'll ever need. You can get it from your dealer for $195 or, as a certified developer, order it through Developer Relations for the amazing price of $75.

Hello, Information?

Do you have a number for the Trap Dispatcher? If you buy a copy of MDS, you'll also receive a copy of the widely rumored yet very real promotional edition of *Inside Macintosh*. This version is essentially a snapshot of what the three-ring binder version contained as of the February Supplement update. However, the promotional

edition is in phone-book format, making it more convenient to transport and use. The part number for this special bundle of MDS/*Inside Macintosh* is M0534, and should be used when ordering from Apple. The final version of *Inside Macintosh* is being published by Addison-Wesley and will be available in bookstores in late summer.

Those of you who simply want copies of the "phone-book" version can get them for $25 from our mailhouse. Send your check or money order to: Apple Computer, Inc., 467 Saratoga Avenue, Suite 621, San Jose, CA 95126. Orders originating from California must include sales tax.

Switcher Hits the Streets

By the time you read this, a number of you will have already received a copy of Switcher 2.0, the first "official pre-release" version of this nifty product. For those who've been in solitary confinement for the past three months, Switcher is a utility program written by Andy Hertzfeld that will allow you to physically partition a Macintosh 512K or a Macintosh XL into a number of smaller logical partitions. You can then load multiple applications into each of the logical partitions and literally switch and transfer data between applications almost instantaneously. *InfoWorld* has described Switcher as a "TopView-like environment that works." (I describe it as the best thing to happen since the transistor.)

We've sent a prerelease copy of Switcher to all registered Macintosh developers. Those of you who haven't received a copy in the mail can download it from CompuServe or pick up a copy from someone who already has it. Please make sure that the current version works properly with your application.

Our plans call for four weeks of testing with the 2.0 version. A subsequent version will be included on the May Software Supplement along with testing instructions and documentation. By mid-June we hope to have a version stable enough to license to all third-party developers. For a reasonable licensing fee, you'll be able to include Switcher on your disk, along with your application.

Softcon Update

A lot of wonderful Macintosh software and hardware from a number of old friends, but nothing else really exciting. The highlight for me? Acquiring a set of IBM luggage tags that read "IBM Application Software," and on the other side (beneath my business card), "When You Think Software Think IBM." Think about that until next month...

◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

Macworld Expo by the Bay

The February Macworld Expo in San Francisco provided us with excitement, enthusiasm, and optimism. The upcoming Macworld Expo in Boston, scheduled for August 21-23 at the Bayside Exposition Center, promises to be even bigger and

better.

As of April 1, exhibitors have signed up for more than 200 booths, and the number is still growing. (In San Francisco, the total was 200.) The list of exhibitors is impressive — including companies like Hayden, Hayes, Living Videotext, Lotus Development, Microsoft, Odesta, Software Arts, and Tecmar. (Wow!) We'll be there, of course — in the Apple booth, giving presentations, and talking to all of you.

As a Macintosh developer, you won't want to miss this major Macintosh event. Macworld Expo will provide you with many opportunities: you can gain an incredible amount of exposure for your products to the press and public, establish important contacts, and meet other developers!

Contact Terry Hamilton at Mitch Hall and Associates for additional information and booth space at (617)329-7466.

◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

Publishing Your Product

By Matt Cobb, Software Promotions Manager

Picture this: You've just spent the last six months of your life creating the next monster megahit Macintosh product. Debugging is going well, and your attention turns to the market. Visions of 18-point dollar signs ($$) float in front of your eyes, just outside your grasp. The question, "How do I cash in?" rings in your ears. This usually translates into another question, "Should I publish myself, or should I go to an established publisher?"

In the Macintosh Developers Group, this is one of the most frequently asked marketing questions; it seems to come up several times a week in the course of phone conversations or visits with developers. There are no clear-cut answers, but this month's column will touch on some of the important issues you should consider in making this major decision.

Publishing Yourself

This is the option that has traditionally been the most popular in our industry. Visions of being the next Lotus seduce many developers into the publishing business. But for every Lotus that soars to the financial stratosphere, there are many that crash and burn, leaving no (financial) survivors. Publishing yourself is clearly the high-risk high-reward path.

To publish yourself in the current market requires a major resource commitment, business savvy, a great product, and a heavy dose of luck and good fortune. If you

choose the self-publish route, be prepared to:

- Design and manufacture high-quality, expensive packaging and collateral material.
- Hire advertising and PR firms for high-impact marketing communications.
- Invest tens of thousands of dollars for major trade show participation.
- Cash in frequent-flyer bonuses accrued from numerous — and expensive — calls on East and West cost distributers.
- Raise a significant amount of capital to fund all of the above.

If your product is truly unique and has little competition in the market, or if it requires nonstandard distribution or support (for example, vertical software), then the points listed above are significantly less severe, and it may be smart to pursue a self-publishing strategy. If not, you may want to consider contacting one of the 50 publishers who are interested in publishing Macintosh products.

Having Someone Else Publish

On the surface, letting someone else publish your product seems like giving away your hard-earned fortune. On the average, royalties from publishers are significantly less than the revenue stream you might get from publishing yourself. But in the end, the risk-adjusted bottom line is likely to be better. Consider the benefits you may get from an established publisher:

- Established contacts with distributors and dealers.
- Brand awareness.
- Experience in advertising, PR, and promotion.
- Capital base and diversified risk.

In addition to these tangible benefits, there are two intangibles that may be even more important. First, if your expertise is as a developer and not as a publisher, over the long term you may be better off concentrating your scarce resources (human and capital) on doing what you know best. Face it — not many people in this world can develop great Macintosh applications. If you can, it's a tremendous wealth-producing advantage you have over others. It makes sense to leave publishing to someone who is an expert in *that* area. Second, over the next few years the publishing business may become dominated by a small number of large publishers, a change forced by the high cost of doing business. This means it will be increasingly difficult for small publishers with one or a few products to survive.

For those who decide that finding a publisher is the smart way to go, we have just finished a resource guide listing Macintosh publishers, which can help in the search for the right partner. In the past few weeks, we mailed this to all the active developers we had on our database. If you didn't receive one, please contact Hazel Holby (MS 4T, 20525 Mariani Avenue, Cupertino, CA or (408) 973-3133) and we'll be happy to mail you a copy. To provide more detailed information, we will feature interviews with three of the top Macintosh publishers starting in next month's issue of *Outside Macintosh*.

◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

Your Font Names Please

By Bill Dawson

In last month's issue of *Outside Macintosh*, the last paragraph of the article "Signatures, Fonts, File Types, and Faith" was inadvertently omitted. Here, then is the section on "Fonts" in its entirety.

...This brings me to *The Dreaded Font ID Number*. Fonts as things exist now are kept separate in the Finder by Font ID Numbers (New York is 2, Geneva is 3, Monaco is 4, ...). Everything up to 255 works. What is wrong with this system? There are more than 255 fonts! Oops. So what do we do now? Fortunately all of these fonts have names. So we can keep track of them by NAME. If you have had any dealings with fonts up to this point you know that they still need a Number to be identified by the Finder. I will continue to assign Font ID Numbers. The change will be that the number I assign you will have already been assigned to someone else — we're wrapping around. I will however, tell you who else has that font # and what the name of it is so you can warn your potential users of installing it on a disk with the same ID. Until further notice this is how the world of Fonts will be.

This brings me still to another grave problem and the problem is mine. As I stated earlier, I've been doing this since I started with Macintosh. Naturally I didn't know as much about Fonts, Creators, and File Types as I do now.. so when I began to keep track of them I only kept account of WHO has WHAT and not WHAT IT WAS they had. I can tell you T-Maker, for instance, has 210-215, but I can't tell you what they call each font. So I have the blues. And I have the task to find out ALL THE NAMES to ALL THE FONTS. I'll be calling each of you in the next few weeks, but it would be a great help for all of us involved if you called me too! Write or anything. If you think there may be ANY problem at all; if you think there may be a misunderstanding; if you think Apple is insane doing this; if you think...TELL ME. I will sit down with engineering and voice your input. Thanks.

I'm reachable through Tymnet @ Supt.Mac or MCI @ MACTECH or Apple Computer, Inc., mail-stop 4-T.