MacDeveloper

The Electronic Magazine for Macintosh™ Developers

Issue #6

1/10/86

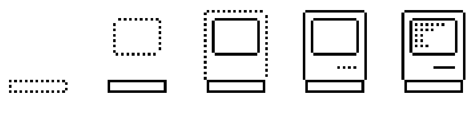
Contents

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 published the first Friday of every other 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 we will get to support the magazine, to everyone's benefit.



In This Issue

In this issue, we get down into the bits.

Using Macros for MDS Trap Calls is the first assembly language article MacDeveloper has published. And it includes a complete listing of the MDS example program "Window," modified to make use of the MDS assembler's macro facilities.

The second article, The PackIt File Format, gives the internal details of the files produced by the program PackIt. This program has become something of a defacto standard for clumping multiple files into one, most often for transport between Macintoshs over telecommunications systems.

Enjoy.

Using Macros for MDS Trap Calls

Laird J. Heal

Introduction

The Macintosh Development System (MDS) has a text-replacement macro facility which can be used to shorten both program listing size and development time, if Toolbox calls are done through such Macros. In this article I introduce macroinstructions, and describe the methods by which they can help a programmer use the Toolbox. As an example I use the "Window" program supplied with MDS and give a listing using macros for almost all of its trap calls.

Macro This and Macro That — an introduction to assembly-language macros

The term "macro", from the Greek word for "larger", is used as a computer term meaning a single action or command which expands itself into many other actions or commands of the same type as the original. Thus, a "macro key" is one that saves typing by taking on the same meaning as typing several characters in a string. In many computer languages, a "macro instruction" is one which can expand itself into many lines of code.

Lately the term "macro" has been used fairly loosely. That is, what are being called "macros" are really one thing that is transformed into another thing. Thus, a macro in many computer languages is a special command which is interpreted by a preprocessor and translates itself into lines of code. Also, "macro keys" often are set up by a special program, and become not just typing savers but function keys that can start new programs or operate conditionally based on a predefined set of criteria.

This is not true with most assembler-language macroinstructions. The functions which define new instructions, or "macros", are themselves part of the assembler language, and the macros themselves have a syntax identical to the other assembler instructions. In effect, the macros translate themselves into other commands of the same language.

Macintosh Development System Macro Advantages and Disadvantages

As described in the *Macintosh™ 68000 Development System User's Manual*, MDS macros come in two flavors. The first is compatible with the Lisa Pascal Workshop assembler. Nine parameters are allowed, and they are named %1 through %9. The other type is called "Macintosh-Style Macros" and uses arguments with string names within the macro. Here, if a parameter is called eventMask in the macro, then {eventMask} tells the assembler to substitute the value passed to the macro in the generated code. This is a very simple text substitution.

There is also an assembly-time IF...THEN...ELSE sequence available in MDS which can help make the macros more general. Since MDS does not allow arguments to be omitted, I often pass an argument of 0 (or NIL, EQUated to 0), and use an IF sequence to test for that value. This is particularly useful when the trap call has a return value. By testing for a NIL return parameter, I can optionally pop the return value from the stack or leave it there for the program to deal with. If I want to leave the value on the stack, then all I need to do is put NIL in the last parameter, and the stack won't be popped. Of course, the value will not be automatically saved either!

When I first got my Macintosh and the 68000 Development System, I was somewhat overwhelmed by *Inside Macintosh* and the toolbox. One of my biggest problems, aside from putting everything together, was to figure out how to put the right parameters in the right place. The trouble lies in the fact that most (but not all) of the traps conform to Pascal conventions. The programmer must remember the conventions, and he must remember which routines use them and which do not.

The Pascal calling conventions are as follows: first reserve a spot on the stack for the result, then, reading the call from left to right, push each parameter onto the stack. This is followed by either a JSR or an 'A-line trap' instruction. All a routine has to do, then, is save the return address off the stack (pushed by the JSR), and then pop the parameters (right to left in the subroutine), MOVE (rather than push) the result on the stack, and return to the saved return address. The caller will then receive the result on the stack, and can then pop it off to restore the stack pointer (A7) to its starting point.

In additon, it should be noted that Lisa Pascal expects structures and strings generally to be passed by address — except for objects of four or less bytes, which should be put directly on the stack. Parameters of type Point are the most typical example of this. This four-byte structure gets passed by value rather than by address. Of course, any Pascal VAR parameter must be passed by address, because the called subroutine will change the value in the caller's memory area.

Coding the traps can be a somewhat mechanical process, then. As long as the method is known by the programmer, it is certainly reasonable to expect that it can be done correctly. However, as described in *Inside Macintosh*, not all of the trap routines follow the same Pascal convention, and there is always the possibility of typing errors which might not get caught in assembly code the way they do in a "strongly typed" language like Pascal.

Using Macros to Replace Trap Calls.

My solution to this problem was to create a set of macros for all of the traps in *Inside Macintosh*. Around the time I did this, I bought a Lisa (or Macintosh XL), along with the Lisa Pascal Workshop 3.0. I'm no Pascal fan, but I couldn't beat the price, at the time, for this kind of a development machine. I did write a few things in Pascal, to be sure, but after I got a C compiler, I started to use that for any work which was better-suited to a higher-level language. However, one thing the Pascal workshop did for me was to let me inspect the assembly language output of the compiler. I used this to compile every trap I could think of and inspected the output to be sure it did the same thing as my MDS macros.

Naturally, after I was for the most part done, I thought of a better way to accomplish the same end. If I had compiled all of the dummy programs, generating the assembler listing, and then ported that listing over to Macintosh, editing it at some point, I could probably have cut down on the number of dumb errors and also on the time involved. On the other hand, putting the macros together by hand was an excellent way to learn them all!

There are two ways to use these macros directly to assemble code (one could also inspect them as examples, of course). The first way is to INCLUDE the macro data set, in the same way as the system traps and equates. On a 128K Macintosh, the assembler runs out of memory when all the macros are brought in, however. Since I upgraded to 512K, I have not had any problems even when running RamDisks of up to 256-350K, so the memory did not overflow by much, I guess. On the Macintosh XL under MacWorks, with a megabyte of memory, there were no problems.

Another way to use the macros is to place them at the beginning of the program, much as I did with the "Window" program supplied with MDS. As I grouped the macros roughly by *Inside Macintosh* order, I could use Edit to open each file, select the macros I needed, and paste them into the program. This also works fairly well.

MDS has a problem with its macro interpreter sometimes. The most "reliable" problem is that the constant "D0" is not passable. Perhaps there is an improved and fixed version available, but as long as it can be worked around I usually do so first. After all, when working in assembly, the code that is generated is the important thing, and convenience is only a bonus. In cases where I need D0 or something else that won't pass, I just paste in the macro, cut out the conditional stuff, and use it that way. The macros aide in programming the Macintosh, but are not an end in themselves, and do not need to be used **every** time a Toolbox routine is called.

A more serious problem from my point of view is that MDS will not allow a variable argument list in "Macintosh-Style Macros". I have made the last parameter of those traps returning results an optional parameter. When I first wrote them, I used a conditional testing for a value '0' in that parameter to signify omission, but I now feel 'NIL' is a more appropriate choice. At any rate, the macro tests for the parameter's value, and either does a MOVE (A7)+,[result] or does nothing, leaving the result on the stack.

Recoding "Window" with Macros

I first received the pre-release MDS system. The first version of Window I saw, from the pre-release, was somewhat unstructured. It did give a working example of how to access TextEdit from MDS, but did so haphazardly. I coded up my macros, and put them into Window, and then decided to revise it a little, putting in subroutines where they were called for, et cetera. A little while later, the released version of MDS came in, and to my surprise that very thing had been done! Also, many of the storage items had been replaced by "Register variables"; this helped shorten the code as well as make it faster and probably more understandable.

I had told a friend I was going to send him these macros, so I quickly put them into the revised version of Window. I was also trying to put together macros for MacinTalk and AppleTalk, but had other things to do, so wound up stopping at the Toolbox. If I ever have a reason to call either of those packages from MDS, I'll have to put those together.

The Recoded "Window"

Here, finally, is the "Window" program, recoded to use Toolbox call macros:

```
  ; File  MyWindow.Asm
  ;-------------------------------------------------------------------------
  ;      Macintosh 68000 Development System -- Programming Example
  ;-------------------------------------------------------------------------
  ; Taken from Window.Asm, written by Marc Russell Benioff and Ernie Beernink
  ; and supplied with the Macintosh 68000 Development System
  ; Modified to demonstrate macro usage by Laird Heal.
  ; This application displays a window within which you can enter and edit
  ; text. Program control is through three menus: the Apple menu, the File
  ; menu, and the Edit menu.
  ; The Apple menu has the standard desk accessories and an About feature.
  ; The File menu lets you quit the application, or launch Edit, an effective
  ; time-saver when debugging.
  ; The Edit menu lets you cut, copy, paste, and clear the text in the window
  ; or in the desk accessories.  Undo is provided for desk accesories only.
  ; Command key equivalents for undo, cut, copy, and paste are provided.
  ; Cutting and pasting between the application and the desk accessories is
  ; not supported.  This requires use of the Scrap Manager.
  ; This program requires the use of a resource file called "MyWindow.Rsrc"
  ; MyWindow.Rsrc is created from "MyWindow.R" using RMaker
  ;--------------------------------   MACROS   ------------------------------
  ;PROCEDURE   AddResMenu(theMenu: MenuHandle; theType: ResType);        MACRO
AddResMenu theMenu,theType =                   MOVE.L        {theMenu},-(SP)
        MOVE.L  {theType},-(SP)                          _AddResMenu
        |
  ;PROCEDURE   BeginUpdate (theWindow: WindowPtr);          MACRO BeginUpdate
```

```
theWindow =             MOVE.L                  {theWindow},-(SP)       ;Move Window
Pointer to Stack  _BeginUpDate                                          ;Begin the
Update            |
        MACRO  Chain  ChainCB,NameStr,SSBuffers  =  LEA                 {ChainCB},A0
        MOVE.L  {SSBuffers},4(A0)               ; Main sound and screen buffers
PEA           {NameStr}         ; Application Name        MOVE.L (SP)+,0(A0)
; Register based trap     _Launch                                       |
   ;PROCEDURE   DisableItem(theMenu: MenuHandle; item: INTEGER);    MACRO
DisableItem menu,item =                         MOVE.L          {menu},-(SP)
MOVE      {item},-(SP)              _DisableItem                        |
   ;PROCEDURE   DragWindow(theWindow: WindowPtr;
startPt:Point;boundsRect:Rect);                 MACRO           DragWindow
theWindow,startPt,boundsRect =                  MOVE.L          {theWindow},-(SP)
;Push Window Pointer    MOVE.L                  {startPt},-(SP)         ;Push Point
onto stack    PEA {boundsRect}                  ;Push addr(Bounds of Drag)
_DragWindow                                     |
   ;PROCEDURE   EnableItem(theMenu: MenuHandle; item: INTEGER);     MACRO
EnableItem menu,item =                          MOVE.L          {menu},-(SP)
MOVE      {item},-(SP)              _EnableItem                         |
   ;PROCEDURE   EndUpdate (theWindow: WindowPtr);   MACRO EndUpdate theWindow =
            MOVE.L      {theWindow},-(SP)       ;Move Window Pointer to Stack
_EndUpdate              ;End our Update         |
   ;PROCEDURE   EraseRect(r: Rect);                     MACRO EraseRect r =
PEA           {r}                   _EraseRect          |
   ;PROCEDURE   FlushEvents(eventMask,stopMask: INTEGER);  MACRO FlushEvents
eventMask,stopMask =    MOVE.W                  {eventMask},D0                  SWAP
D0                      MOVE.W                  {stopMask},D0
_FlushEvents                      |
   ;PROCEDURE   GetItem(theMenu:MenuHandle;item:INTEGER;VAR
itemString:Str255);     MACRO                   GetItem theMenu,item,itemString =
MOVE.L  {theMenu},-(SP)                         MOVE  {item},-(SP)              ;Item
Number  PEA{itemString}                         ;push string address    _GetItem
                  |
   ;FUNCTION     GetMenu(resourceID: INTEGER): MenuHandle;  MACRO GetMenu
menu_ID,MenuHandle =  CLR.L  -(SP)                              ;space for result
MOVE      {menu_ID},-(SP)       ;Identify Menu 1  _GetRMenu                     ;Get
Menu Handle  IF           {MenuHandle} <> 0             MOVE.L (SP)+,{MenuHandle}
;ELSE Leave Result on Stack   ENDIF                                             |
   ;FUNCTION     GetNewDialog(dialogID: INTEGER; dStorage: Ptr;
   ;                          behind: WindowPtr): DialogPtr;   MACRO GetNewDialog
dialogID,dStorage,behind,DialogPtr =            CLR.L  -(SP)                    ;Clear
Space for Dialog Pointer          MOVE          {dialogID},-(SP)        ;Menu Resource
ID        PEA       {dStorage}                  ;Storage Area           MOVE.L
{behind},-(SP)          _GetNewDialog                           IF
{DialogPtr} <> 0        MOVE.L                  (SP)+,{DialogPtr}               ENDIF
                        |
   ;FUNCTION     GetNewWindow(windowID: INTEGER; wStorage: (*Ptr*) WindowPeek;
   ;                          behind: WindowPtr): WindowPtr;        MACRO
GetNewWindow  windowID,wStorage,behind,PtrResult =          CLR.L   -(SP)
        ;Clear space for result                 MOVE.W          {windowID},-(SP)
        PEA       {wStorage}                    ;  ***Address***  of  a
WindowStorage Record  MOVE.L                    {behind},-(SP)
_GetNewWindow                     IF                    {PtrResult} <> 0
MOVE.L  (SP)+,{PtrResult}                       ENDIF
|
   ;FUNCTION     GetNextEvent(eventMask: INTEGER;
   ;                      VAR theEvent: EventRecord): BOOLEAN;          MACRO
```

```
GetNextEvent eventMask,theEvent,BOOLResult =          CLR.W   -(SP)
;Clear space for event result   MOVE            {eventMask},-(SP)      ;Mask which
events    PEA       {theEvent}                   ;Push addr(event record) on the stack
          _GetNextEvent                          ;Get the next event      IF
{BOOLResult} <> 0             MOVE.B             (SP)+,{BOOLResult}              ENDIF
                              |
   ;PROCEDURE   GlobalToLocal (VAR pt:Point);          MACRO GlobalToLocal pt =
          PEA       {pt}                         ;Mouse Point    _GlobalToLocal
              |
   ;PROCEDURE   InitDialogs(restartProc:  ProcPtr);     MACRO InitDialogs  restartProc
=             IF            {restartProc} = 0            CLR.L    -(SP)
          ELSE                                          MOVE.L {restartProc},-(SP)
          ENDIF                                         _InitDialogs
          |
   ;PROCEDURE   InitGraf(globalPtr:  QDPtr);             MACRO InitGraf  globalPtr =
          PEA       {globalPtr}                         _InitGraf
          |
   ;PROCEDURE   InsertMenu(theMenu: MenuHandle; beforeID: INTEGER);          MACRO
InsertMenu theMenu,beforeID =                 MOVE.L          {theMenu},-(SP)
          MOVE.W {beforeID},-(SP)                       _InsertMenu
          |
   ;PROCEDURE   LocalToGlobal(VAR pt: Point);            MACRO LocalToGlobal pt =
          PEA       {pt}                                _LocalToGlobal
          |
   ;FUNCTION     NewMenu(menuID: INTEGER; menuTitle: Str255): MenuHandle;
MACRO   NewMenu menu_ID,menuTitle,MenuHandle =      CLR.L    -(SP)
;space for result  MOVE  {menu_ID},-(SP)         ;Identify Menu  1         PEA
{menuTitle}   _NewMenu                       IF              {MenuHandle} <> ''
          MOVE.L  (SP)+,{MenuHandle}             ;ELSE Leave Result on Stack    ENDIF
                              |
   ;FUNCTION     OpenResFile(fileName: Str255): INTEGER;       MACRO OpenResFile
fileName,INTResult =      CLR.W -(SP)                          ; Space on Stack for
result    PEA       {fileName}                          _OpenResFile
          IF        {INTResult} <> 0                    MOVE    (SP)+,{INTResult}
          ENDIF                                         |
   ;PROCEDURE   SelectWindow (theWindow: WindowPtr);        MACRO SelectWindow
theWindow =               MOVE.L                 {theWindow},-(SP)              ;Get
Window Pointer to Stack                      _SelectWindow
;Activate  Window         |
   ;PROCEDURE   SetPort  (gp:  GrafPort);                MACRO SetPort gp =
MOVE.L   {gp},-(SP)              ;Get Window Pointer to Stack      _SetPort
              |
   ;PROCEDURE   SystemClick(theEvent: EventRecord; theWindow: WindowPtr);
MACRO   SystemClick theEvent,theWindow =       PEA              {theEvent}
;Put Event Record On Stack      MOVE.L           {theWindow},-(SP)      ;Move Window
Pointer On Stack  _SystemClick                          ;Click  In  System
Window   |
   ;FUNCTION     SystemEdit(editCmd:INTEGER): BOOLEAN;       MACRO SystemEdit
editCmd,SystemEdit =      CLR                   -(SP)                  ;Space for
Problem Result    MOVE  {editCmd},-(SP)          ;System  Action         _SysEdit
              ;Do it        MOVE.B             (SP)+,{SystemEdit}     ;Pop Problem
Result    |
   ;PROCEDURE   SystemTask;
   ;FUNCTION     SystemEvent(theEvent: EventRecord): BOOLEAN;        MACRO
SystemEvent theEvent,BOOLResult =              CLR.W -(SP)                    ;Space
for Problem Result        MOVE.L             {theEvent}
_SystemEvent                                   MOVE.B          (SP)+,{BOOLResult}
```

```
;Pop Problem Result     |
   ;PROCEDURE   TEActivate (hTE: TEHandle);           MACRO TEActivate hTE =
        MOVE.L  {hTE},-(SP)                     ;Move Text Handle to Stack
_TEActivate             ;Activate Text          |
   ;PROCEDURE   TEClick(pt: Point; extend: BOOLEAN; hTE: TEHandle);  MACRO TEClick
pt,extend,hTE =         MOVE.L                  {Pt},-(SP)              ;Mouse Point
(GTL)   IF      {extend} = 0                          CLR             -(SP)
        ;the 68000 wants the SP even            ELSE
MOVE.B  {extend},-(SP) ;push {extend} onto top byte and pad    ENDIF
            MOVE.L     {hTE},-(SP)                    ;Identify Text   _TEClick
                       |
   ;PROCEDURE   TECopy (hTE: TEHandle);               MACRO TECopy hTE =
MOVE.L  {hTE},-(SP)          ;Identify Text     _TECopy
;Copy Text  |
   ;PROCEDURE   TECut (hTE: TEHandle);          MACRO         TECut hTE =
            MOVE.L     {hTE},-(SP)                    ;Identify Text   _TECut
            ;Cut it     |
   ;PROCEDURE   TEDeActivate (hTE: TEHandle).         MACRO TEDeActivate hTE =
        MOVE.L  {hTE},-(SP)                     ;Get Text Handle        _TeDeActivate
            ;Un Activate Text                   |
   ;PROCEDURE   TEDelete(hTE: TEHandle);        MACRO         TEDelete hTE =
        MOVE.L  {hTE},-(SP)                     ;Identify Text _TEDelete
        ;Paste   |
   ;PROCEDURE   TEInsert(text: Ptr; length: LONGINT; hTE: TEHandle); MACRO
TEInsert text,length,hTE =                      MOVE.L        {text},-(SP)
        MOVE.L  {length},-(SP)                        MOVE.L {hTE},-(SP)
        _TEInsert                                     |

   ;PROCEDURE   TEIdle(hTE: TEHandle);                MACRO           TEIdle hTE =
            MOVE.L     {hTE},-(SP)                    ;Push our text handle    _TEIdle
                ;Flash cursor in text window    |
   ;PROCEDURE   TEKey (key: CHAR; hTE: TEHandle);    MACRO TEKey key,hTE =
        MOVE    {key},-(SP)                     ;Get Character          MOVE.L {hTE},-
(SP)        ;Identify Text     _TEKey                                   ;Put Character
InAnd Print it     |
   ;FUNCTION    TENew(destRect,viewRect: Rect): TEHandle;   MACRO TENew
destRect,viewRect,HandleResult =                CLR.L  -(SP)                  ;Clear
Space for TEHandle      PEA                     {destRect}                    PEA
        {viewRect}                              _TENew
        MOVE.L  (SP)+,{HandleResult}                  |
   ;PROCEDURE   TEPaste (hTE: TEHandle);              MACRO TEPaste hTE =
MOVE.L  {hTE},-(SP)          ;Identify Text     _TEPaste                      ;Paste
|
   ;PROCEDURE   TEUpdate(rUpdate: Rect; hTE: TEHandle);       MACRO TEUpdate
rUpdate,hTE =           PEA                     {rUpdate}               ;Put View Rect
On Stack    MOVE.L     {hTE},-(SP)                    ;Identify Text Handle
_TEUpdate               ;Update our Window      |
   ;--------------------------------   INCLUDES   ------------------------------
        Include  MacTraps.D                     ; Use System and ToolBox traps
   Include    ToolEqu.D       ; Use ToolBox equates
   ;----------------------------  Use   of   Registers  --------------------------
   ; Operating System and Toolbox calls always preserve D3-D7, and A2-A4.
   ; Register use: A5-A7 are reserved by the system
   ;             D1-D3, A0-A1 are unused
   ;             D0 is used as a temp
   ModifyReg     EQU   D4   ; D4 holds modifier bits from GetNextEvent
   MenuReg       EQU   D5   ; D5 holds menu ID from MenuSelect,MenuKey
```

```
MenuItemReg  EQU   D6   ; D6 holds item ID from MenuSelect,MenuKey
AppleHReg    EQU   D7   ; D7 holds the handle to the Apple Menu
TextHReg     EQU   A2   ; A2 is a handle to the TextEdit record
WindowPReg   EQU   A3   ; A3 is a pointer to the editing window
EditHReg     EQU   A4   ; A4 is a handle to the Edit menu
;--------------------------------   EQUATES   -------------------------------
; These are equates associated with the resources ; for the Window example.
AppleMenu    EQU   1    ; First item in MENU resource
AboutItem    EQU   1    ; First item in Apple menu
FileMenu     EQU   2    ; Second item in MENU resource
LaunchItem   EQU   1    ; First item in File menu
QuitItem     EQU   2    ; Second item in File menu
EditMenu     EQU   3    ; Third item in MENU resource
UndoItem     EQU   1    ; Items in Edit menu
CutItem      EQU   3    ; (Item 2 is a line)
CopyItem     EQU   4
PasteItem    EQU   5
ClearItem    EQU   6
AboutDialog  EQU   1    ; About dialog is DLOG resource #1
ButtonItem   EQU   3    ; DITL returned by DLOG #1 - changed from source
provided
ASample      EQU   1    ; Sample Window is WIND resource #1
; These are modifier bits returned by the GetNextEvent call.
activeBit    EQU   0    ; Bit position of de/activate in Modify
cmdKey       EQU   8    ; Bit position of command key in Modify
shiftKey     EQU   9    ; Bit position of shift key in Modify
;---------------------------------   XDEFs   --------------------------------
; XDEF all labels that are to be symbolically displayed by MacDB.    XDEF
Start     XDEF          InitManagers           XDEF          OpenMyRsrc
      XDEF          SetupMenu                  XDEF          SetupWindow
      XDEF          SetupTextEdit              XDEF          Activate
      XDEF          Deactivate                 XDEF          Update
XDEF      KeyDown       XDEF                   MouseDown            XDEF
      Sys_Event         XDEF                   Content              XDEF
      Drag          XDEF                       InMenu        XDEF
About
;------------------------------   Main   Program   ----------------------------
Start
      ILLEGAL                           ; take your pick of how to       BRA
*-0                           ; trap to the debugger
;     _Debugger                         ; there's even a trap for it     BSR
InitManagers          ; Initialize managers    BSR   OpenMyRsrc
; Open the resource file          BSR          SetupMenu                  ; Build
menus, draw menu bar              BSR          SetupWindow                ; Draw
Editing Window     BSR          SetupTextEdit                         ; Initialize
TextEdit
EventLoop             ; MAIN PROGRAM LOOP
      _SystemTask                       ; Update Desk Accessories
      TEIdle       TextHReg             ; blink cursor etc.
AllEventMask          EQU       #$0FFF             GetNextEvent
AllEventMask,EventRecord,0
; Look for an event, leave result on stack      CMPI           #0,(SP)+
      ; Test low byte & pop stack   BEQ            EventLoop
; No event... Keep waiting        BSR                 HandleEvent
; Go handle event  BEQ               EventLoop                           ; Not Quit, keep
going          RTS                                ; Quit, exit to Finder    ; Note:
When an event handler finishes, it returns with Z asserted,  ;        unless Quit was
```

```
selected, when it returns with Z negated.  An ;        RTS now will return to the
Finder.
;------------------------------   InitManagers   ----------------------------
InitManagers
;       PEA             -4(A5)                  ; Quickdraw's global area
;       _InitGraf                               ; Init Quickdraw        InitGraf
-4(A5)                  ; Init Quickdraw at its global area
        _InitFonts                              ; Init Font Manager
;       MOVE.L          #$0000FFFF,D0           ; Flush all events
        _FlushEvents    FlushEvents  #$0000,#$FFFF
        _InitWindows                            ; Init Window Manager
_InitMenus                              ; Init Menu Manager
        InitDialogs   0                         ; Init Dialog Manager, no restart
procedure
        _TEInit                                 ; Init Text Edit        _InitCursor
                        ; Turn on arrow cursor          RTS
;------------------------------   OpenMyRsrc   ----------------------------
OpenMyRsrc                                              ; For development, we are
keeping the resources in a separate file.  Otherwise,; use RMaker to bind the
resources with the application, and make the ; OpenResFile call unneccessary. Note:
the explicit volume reference   ; ("ASM1:") is probably bad style; a good exercise
is to make the access ; independent of volume name.  Barring that, it must be done
as it was or  ; the system will search the wrong volume for the file.
        OpenResFile  'ASM1:MyWindow.Rsrc',0 ; name of file, no result          MOVE
        (SP)+,D0                ; Discard refNum        RTS
;------------------------------   SetupMenu   ----------------------------
SetupMenu                                               ; The names of all the menus
and the commands in the menus are stored in the         ; resource file.  The way you
build a menu for an application is by reading   ; each menu in from the resource file
and then inserting it into the current          ; menu bar.  Desk accessories are
read from the system resource file and          ; added to the Apple menu.
; Apple Menu Set Up.
        GetMenu         #AppleMenu,0                    MOVE.L
(SP),AppleHReg          ; Save for later comparison ; MOVE.L            (SP),-(SP)
                ; Copy handle for AddResMenu
; PROCEDURE InsertMenu (menu:MenuHandle; beforeID: INTEGER);            CLR
-(SP)                   ; Append to menu                _InsertMenu
        ; Which is currently empty
; Add Desk Accessories Into Apple menu
        AddResMenu AppleHReg,#'DRVR'
; File Menu Set Up
        GetMenu         #FileMenu,0                     CLR             -(SP)
        ; Append to list  _InsertMenu                                   ; After Apple
menu
; Edit Menu Set Up
        GetMenu         #EditMenu,0                     MOVE.L (SP),EditHReg
; Save for later - leave on stack for InsertMenu        CLR             -(SP)
        ; Append to list  _InsertMenu                                   ; After File
menu            _DrawMenuBar                            ; Display the menu bar  RTS
;-----------------------------   SetupWindow   ----------------------------
SetupWindow                                             ; The window parameters are
stored in our resource file.  Read them from  ; the file and draw the window, then
set the port to that window.  Note that         ; the window parameters could just as
easily have been set using the call   ; NewWindow, which doesn't use the resource
file.
        GetNewWindow  #ASample,WindowStorage(A5),#-1,0          MOVE.L
        (SP),WindowPReg         ; Save for later
```

```
; PROCEDURE SetPort (gp: GrafPort)          ; Pointer still on stack
_SetPort                          ; Make it the current port        RTS
;----------------------------   SetupTextEdit   --------------------------
SetupTextEdit                                        ; Create a new text record for
TextEdit, and define the window within which; it will be displayed.  Note that if the
window boundaries are changed in   ; the resource file, DestRect and ViewRect will
have to be changed too.
      TENew        DestRect,ViewRect,TextHreg       RTS
;------------------------   Event   Handling   Routines   ----------------------
HandleEvent                                          ; Use the event number as an
index into the Event table.  These 12 events   ; are all the things that could
spontaneously happen while the program is     ; in the main loop.
      MOVE         Modify,ModifyReg           ; More useful in a reg  MOVE
What,D0               ; Get event number      ADD            D0,D0
; *2 for table index    MOVE         EventTable(D0),D0        ; Point to routine offset
      JMP          EventTable(D0)             ; and jump to it
EventTable
      DC.W         NextEvent-EventTable       ; Null Event (Not used)          DC.W
      MouseDown-EventTable    ; Mouse Down         DC.W            NextEvent-
EventTable    ; Mouse Up (Not used)          DC.W            KeyDown-EventTable
; Key Down   DC.W     NextEvent-EventTable  ; Key Up (Not used)      DC.W
KeyDown-EventTable  ; Auto Key    DC.W            Update-EventTable        ;
Update        DC.W     NextEvent-EventTable  ; Disk (Not used)        DC.W
Activate-EventTable  ; Activate    DC.W            NextEvent-EventTable  ; Abort
(Not used)    DC.W     NextEvent-EventTable  ; Network (Not used)   DC.W
NextEvent-EventTable            ; I/O Driver (Not used)
;---------------------------   Event   Actions   ---------------------------
Activate                                             ; An activate event is posted
by the system when a window needs to be      ; activated or deactivated.  The
information that indicates which window       ; needs to be updated was returned by
the NextEvent call.
      CMP.L        Message,WindowPReg       ; Was it our window?  BNE
NextEvent             ; No, get next event     BTST            #ActiveBit,ModifyReg
; Activate?   BEQ      Deactivate                    ; No, go do Deactivate   ; To
activate our window, activate TextEdit, and disable Undo since we don't       ;
support it.  Then set our window as the port since an accessory may have    ;
changed it.  This activate event was generated by SelectWindow as a result  ; of a
click in the content region of our window.  If the window had scroll   ; bars, we
would do ShowControl and HideControl here too.
      TEActivate   TextHreg         ; Activate what this TextHandle points to
      DisableItem  EditHReg,#UndoItem
SetOurPort                                     ; used by InAppleMenu
      SetPort          WindowPReg             ; set the port for our control
NextEvent
      MOVEQ            #0,D0                  ; Say that it's not Quit          RTS
                                ; return to EventLoop
Deactivate                                           ; To deactivate our window,
turn off TextEdit, and Enable undo for the desk        ; accessories (which must be
active instead of us).
      TeDeActivate    TextHReg      ; Deactivate what this TextHandle points to
      EnableItem       EditHReg,#UndoItem             BRA
NextEvent          ; Go get next event
Update                                               ; The window needs to be
redrawn.  Erase the window and then call TextEdit   ; to redraw it.
      BeginUpdate      WindowPReg             ; point to the window to update
      EraseRect        ViewRect               ; erase inside the rect
      TEUpdate         ViewRect,TextHReg
```

```
	EndUpdate	WindowPReg		; tell Quickdraw we are done for now
BRA		NextEvent	; Go get next event
KeyDown						; A key was pressed.  First
check to see if it was a command key.  If so,  ; go do it.  Otherwise pass the key to
TextEdit.
	BTST		#CmdKey,ModifyReg	; Is command key down?		BNE
		CommandDown	; If so, handle command key
	TEKey		Message+2,TextHReg		BRA
NextEvent		; Go get next event

 CommandDown						; The command key was down.
Call MenuKey to find out if it was the command		; key equivalent for a menu
command, pass the menu and item numbers to Choices.
; FUNCTION  MenuKey (ch:CHAR): LongInt;		CLR.L			-(SP)
		; Space for Menu and Item		MOVE			Message+2,-
(SP)  ; Get character  _MenuKey					; See if it's a
command	MOVE			(SP)+,MenuReg  ; Save Menu	MOVE
	(SP)+,MenuItemReg		; and Menu Item		BRA			Choices
		; Go dispatch command

;--------------------Mouse  Down  Events  And  Their  Actions--------------------
--
MouseDown						; If the mouse button was
pressed, we must determine where the click  ; occurred before we can do anything.
Call FindWindow to determine	; where the click was
; dispatch the event according to the result.
; FUNCTION  FindWindow (thePt: Point;		;				VAR
whichWindow: WindowPtr): INTEGER;		CLR		-(SP)
; Space for result  MOVE.L		Point,-(SP)			; Get mouse
coordinates  PEA  WWindow			; Event Window		_FindWindow
			; Who's got the click?  MOVE		(SP)+,D0
	; Get region number		ADD			D0,D0			; *2
for index into table	MOVE		WindowTable(D0),D0	; Point to routine offset
	JMP		WindowTable(D0)		; Jump to routine
 WindowTable
	DC.W		NextEvent-WindowTable	; In Desk (Not used)	DC.W
InMenu-WindowTable  ; In Menu Bar		DC.W		Sys_Event-
WindowTable		; System Window		DC.W		Content-WindowTable
; In Content  DC.W	Drag-WindowTable	; In Drag		DC.W
NextEvent-WindowTable		; In Grow (Not used)		DC.W
NextEvent-WindowTable		; In Go Away (Not used)
Sys_Event						; The mouse button was
pressed in a system window.  SystemClick calls the ; appropriate desk accessory to
handle the event.
	SystemClick EventRecord,WWindow	; This is not our window, let the
owner handle it	BRA			NextEvent			; Go get next
event
Content							; The click was in the content
area of a window.  If our window was in		; front, then call Quickdraw to get
local coordinates, then pass the ; coordinates to TextEdit. We also determine
whether the shift key was		; pressed so TextEdit can do shift-clicking. If our
window wasn't in  ; front, move it to the front, but don't process click.
	CLR.L		-(SP)				; clear room for result
_FrontWindow				; get FrontWindow		MOVE.L
(SP)+,D0			; Is front window pointer		CMP.L
WindowPReg,D0		; same as our pointer?  BEQ.S		@1
	; Yes, call TextEdit		; We weren't active, select our window.  This
```

```
causes an activate event.
      SelectWindow   WWindow                  BRA
NextEvent        ; and get next event
@1                                            ; We were active, pass the
click (with shift) to TextEdit.
      GlobalToLocal   Point
      BTST       #shiftKey,ModifyReg      ; Is shift key down?    SNE
D0                   ; True if shift down
; PROCEDURE      TEClick (pt: Point; extend: BOOLEAN; hTE: TEHandle);     MOVE.L
      Point,-(SP)               ; Mouse Point (GTL)         ; Note:  We want the
boolean in the high byte, so use MOVE.B.  The 68000 ; pushes an extra, unused byte
on the stack for us.
      MOVE.B         D0,-(SP)                 MOVE.L         TextHReg,-
(SP)        ; Identify Text     _TEClick                               ;
TEClick
;     TEClick      Point,D0,TextHReg        ; unfortunately doesn't assemble!
BRA          NextEvent                    ; Go get next event
Drag
; The click was in the drag bar of the window.  Draggit.
      DragWindow WWindow,Point,Bounds         BRA            NextEvent
             ; Go get next event     InMenu
      ; The click was in the menu bar.  Determine which menu was selected, then
; call the appropriate routine.
; FUNCTION  MenuSelect (startPt:Point) : LongInt;   CLR.L          -(SP)
      ; Get Space For Menu Choice  MOVE.L           Point,-(SP)
; Mouse At Time Of Event        _MenuSelect                            ; Menu
Select       MOVE                (SP)+,MenuReg             ; Save Menu    MOVE
      (SP)+,MenuItemReg        ; and Menu Item      ; On entry to Choices, the
resource ID of the Menu is saved in the low   ; word of a register, and the resource
ID of the MenuItem in another.   ; The routine MenuKey, used when a command key
is pressed, returns the same    ; info.
Choices                                       ; Called by command key too
      CMP          #AppleMenu,MenuReg       ; Is It In Apple Menu?  BEQ
InAppleMenu          ; Go do Apple Menu      CMP             #FileMenu,MenuReg
; Is It In File Menu?    BEQ            InFileMenu                     ; Go do File
Menu         CMP      #EditMenu,MenuReg     ; Is It In Edit Menu?     BEQ
InEditMenu           ; Go do Edit Menu        ChoiceReturn
      BSR          UnHiliteMenu             ; Unhighlight the menu bar      BRA
      NextEvent                 ; Go get next event
InFileMenu
; Add the following code to check for Edit transfer request
      CMP    #LaunchItem,MenuItemReg       ; Is It Edit?     BEQ
Launch_Edit          ; Yes, Launch him

; If it was in the File menu, just check for Quit since that's all there is.
      CMP          #QuitItem,MenuItemReg   ; Is It Quit?     BNE
ChoiceReturn         ; No, Go get next event        BSR            UnHiliteMenu
             ; Unhighlight the menu bar         MOVE           #-1,D0
; say it was Quit   RTS
InEditMenu                                    ; First, call SystemEdit.  If a
desk accessory is active that uses the Edit    ; menu (such as the Notepad) this lets
it use our menu.    ; Decide whether it was cut, copy, paste, or clear.  Ignore Undo
since we      ; didn't implement it.
      BSR          sys_edit                 ; Desk accessory active?       BNE.S
      ChoiceReturn                ; Yes, sys_edit handled it        CMP
#CutItem,MenuItemReg          ; Is It Cut?   BEQ             Cut
; Yes, go handle it CMP           #CopyItem,MenuItemReg         ; Is it Copy?
```

```
BEQ         Copy                ; Yes, go handle it   CMP
#PasteItem,MenuItemReg          ; Is it Paste?        BEQ          Paste
      ; Yes, go handle it       CMP                   #ClearItem,MenuItemReg
; Is it Clear?      BEQ         ClearIt               ; Yes, go handle it
BRA.S               ChoiceReturn          ; Go get next event
InAppleMenu                                           ; It was in the Apple menu.  If it
wasn't About, then it must have been a    ; desk accessory.  If so, open the desk
accessory.
      CMP           #AboutItem,MenuItemReg ; Is It About?  BEQ          About
            ; If So Goto About...
; PROCEDURE GetItem (menu: MenuHandle; item: INTEGER;    ;               VAR
itemString: Str255); ;          MOVE.L                AppleHReg,-(SP)       ; Look
in Apple Menu ;     MOVE              MenuItemReg,-(SP)      ; What Item Number? ;
PEA         DeskName                      ; Get Item Name ;       _GetItem
                    ; Get Item     GetItem   AppleHReg,MenuItemReg,DeskName
; FUNCTION  OpenDeskAcc (theAcc: Str255) : INTEGER;       CLR            -(SP)
            ; Space For Opening Result    PEA           DeskName
      ; Open Desk Acc           _OpenDeskAcc                             ; Open
It    MOVE          (SP)+,D0              ; Pop result
GoSetOurPort
      BSR           SetOurPort            ; Set port to us        BRA.S
ChoiceReturn        ; Unhilite menu and return        ;---------------------------
Text  Editing  Routines  -----------------------
Cut                             ; CUT
      TECut         TextHReg                          BRA          ChoiceReturn
            ; Go get next event
Copy                            ; COPY
      TECopy        TextHReg                          BRA          ChoiceReturn
            ; Go get next event
Paste                           ; PASTE
      TEPaste       TextHReg                          BRA          ChoiceReturn
            ; Go get next event
ClearIt                                   ;CLEAR
      TEDelete      TextHReg                          ; Clear without copying
                    BRA               ChoiceReturn                 ; Go get next
event; SystemEdit does undo, cut, copy, paste, and clear for desk accessories.
; It returns False (BEQ) if the active window doesn't belong to a      ; desk
accessory.
sys_edit
; FUNCTION  SystemEdit (editCmd:INTEGER): BOOLEAN;       CLR            -(SP)
            ; Space for result  MOVE                  MenuItemReg,-(SP)     ; Get
item in Edit menu   SUBQ              #1,(SP)                ; SystemEdit is off by
1     _SysEdit                            ; Do It          MOVE.B
(SP)+,D0            ; Pop result  RTS
; BEQ if NOT handled
UnhiliteMenu
; PROCEDURE HiLiteMenu (menuID: INTEGER);         CLR          -(SP)
      ; All Menus   _HiLiteMenu                          ; UnHilite Them All
      RTS                                             ;---------------------------
------Misc   Routines--------------------------
About                                                 ; Call GetNewDialog to read the
dialog box parameters from the resource file ; and display the box.  Set the port to
the box, then wait for the proper   ; click or keypress.  Finally, close the dialog box
and set the pointer to us.
      GetNewDialog    #AboutDialog,DStorage,#-1,0          MOVE.L         (SP),-
(SP)                ; Copy handle for Close
; PROCEDURE SetPort (gp: GrafPort)        ; Handle already on stack
```

```
_SetPort                          ; Make dialog box the port
       TEDeActivate   TextHReg                ; Deactivate Text
WaitOK
; PROCEDURE ModalDialog (filterProc: ProcPtr;        ;                              VAR
itemHit: INTEGER);     CLR.L          -(SP)                     ; Clear space For
handle        PEA      ItemHit                   ; Storage for item hit  _ModalDialog
                          ; Wait for a response
       MOVE         ItemHit,D0                   ; Look to see what was hit       CMP
       #ButtonItem,D0              ; was it OK?  BNE            WaitOK
; No, wait for OK       ; PROCEDURE CloseDialog (theDialog: DialogPtr);
_CloseDialog                      ; Handle already on stack           BRA
GoSetOurPort            ; Set port to us and return       Launch_Edit
                ; Launch the Editor(Edit MyWindow.Asm)
                      ; see Segment Loader manual       Chain
LaunchCB(A5),'Upper:Edit',#0

;  ----------------------------  Data   Starts   Here   -------------------------
EventRecord                       ; NextEvent's Record  What:      DC      0
                      ; Event number  Message: DC.L   0
; Additional information  When: DC.L          0                            ; Time
event was posted   Point:        DC.L          0                            ; Mouse
coordinates  Modify:  DC        0                         ; State of keys and
button  WWindow: DC.L           0                         ; Find Window's Result
DStorage                                                DCB.W  DWindLen,0
; Storage For Dialog DeskName  DCB.W        16,0                            ; Desk
Accessory's Name Bounds       DC            28,4,308,508; Drag Window's Bounds
ViewRect      DC       5,4,245,405; Text Record's View Rect DestRect       DC
       5,4,245,405    ; Text Record's Dest Rect ItemHit      DC             0
                ; Item clicked in dialog
;------------------------   Nonrelocatable   Storage   ----------------------
; Variables declared using DS are placed in a global space relative to ; A5.  When
these variables are referenced, A5 must be explicitly mentioned.
WindowStorage      DS.W          WindowSize  ; Storage for Window LaunchCB
DS.l          2                          ;Used to _Launch to Editor
End
```

Sources for Macintosh Trap Macros.

I went through *Inside Macintosh* and drafted MDS macros for each of the traps, excluding AppleTalk and most of the _Control calls. When the Macintosh XL (Lisa 2/10) became affordable, I double-checked these macros against the code generated by the Lisa Pascal Workshop, and did uncover a lot of my mistakes! (I'm still running into them, every so often.) On the other hand, the Pascal compiler can generate some pretty odd code for a trap, too (Munger() comes to mind).

In order for a reader to produce a set of these on his own, I would advise doing it one of the easy ways, rather than going through *Inside Macintosh* and coding up each one (although there is no better way to become acquainted with all of the traps, it takes a couple weeks easy, just in the typing). For one thing, MacNosy provides a set of routines containing all of the Lisa Pascal "glue" to disassemble. For another, the Pascal compiler itself generates the trap calls directly and can be directed to generate (Lisa Workshop) assembler code. Dummy programs containing the trap calls can then be inspected and briefly edited.

I did do this a while ago however, basically for my own work. I have given the source to these macros, along with Lisa Pascal-generated listings, to a few friends, and I have no intention of restricting any <u>non-commercial</u> distribution of this source. If anyone wants a diskette copy of these macros from me, however, I am asking for $10 to cover costs (and to interest me in copying the disk and mailing it out); or send me SASE, disk and $5.

If you are interested in this, or if you have any comments about this article or its subject matter, send mail to:

Laird J. Heal
P. O. Box 1485
Salem, NH 03079

The PackIt File Format

Harry R. Chesley

Introduction

This article describes the packed file format (protocol) used by the PackIt program [Note: PackIt was written by the author]. PackIt has found fairly wide acceptance in the Macintosh community, and wider publication of the file format used seems appropriate — this document, in an earlier incarnation, has been available essentially since PackIt itself first became available. It is also being presented as a possible informal standard; details on how it might work in the context of other existing standards, and how it might be expanded in the future to include such features as file compression and encryption are described later in the article.

The goals of the PackIt protocol are:

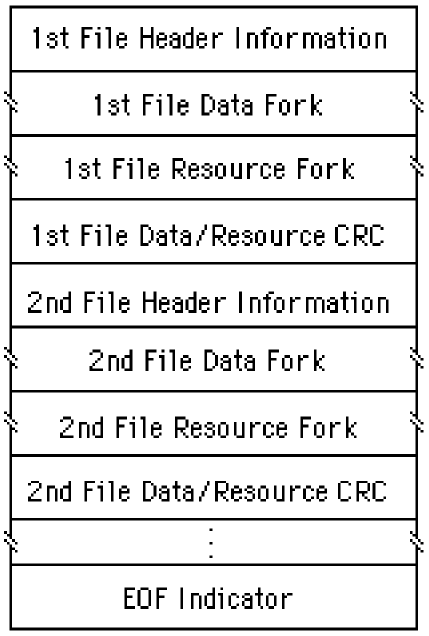
- Allow multiple files to be transported as a single file.
- Preserve both forks of the file, as well as appropriate header information.
- Provide a strong integrity verification to ensure that the files are intact, and also that unpacking them will in no way damage the file system.
- Allow future expansion and extention.

Specifically discluded from PackIt's goals are:

- Transport of the files between the packing and the unpacking sites.
- Error correction (as distinct from detection).
- Data compression (at this time).

Format

The file format used by PackIt is quite straight-forward:


1st File Header Information
1st File Data Fork
1st File Resource Fork
1st File Data/Resource CRC
2nd File Header Information
2nd File Data Fork
2nd File Resource Fork
2nd File Data/Resource CRC
EOF Indicator

Each sub-file consists of a block of header information, the contents of the file's data fork, the resource fork, and a CRC over the data and resource forks. Following all of the sub-files is an end-of-file indicator.

The header information has the following format:

```
struct packFHead {           /* Packed file info header format. */
        long magic;          /* Magic number. */
        char fName[64];      /* Suggested file name. */
        OSType fType;        /* File type. */
        OSType fCreator;     /* File creator. */
        short fFlags;        /* File (finder) flags. */
        int fLocked;         /* File locked if non-zero. */
        long DSize;          /* Data fork size (bytes). */
        long RSize;          /* Resource fork size (bytes). */
        long fCrDate;        /* File creation date. */
        long fModDate;       /* File modification date. */
        unsigned hCrc;       /* Header CRC. */
};
```

The magic number is 'PMag'. This should also be thought of as a sub-file format type; it may be used in the future to identify a different packing scheme, allowing different packing schemes to be used within the same packed file. The name, type, creator, finder flags, locked flag, data fork size, resource fork size, creation date, and modification date are all the standard information used in other file transfer protocols, and are described in the File Manager section of *Inside Macintosh*. Although all of the finder flag bits are sent, the initial implementation of PackIt masks off all but "invisible" and "has bundle" when unpacking.

The CRC is taken across the entire header, including the magic number. It allows the unpacker to verify the header before attempting to create a file using that information. The CRC algorithm is given later.

The EOF indicator is 'PEnd'. It allows the unpacker to verify that all files were received, that the file was not truncated during transmission. Most unpackers will want to allow unpacking of the part that did come across in this event, but they should notify the user that not everything is there. This can be especially important with programs that depend on accompanying data files.

The CRC is generated (and checked) using the following algorithm (it's CRC-CCITT):

```
/*    Return the CRC across each byte in the block pointed to by blk, of
size sz. Start with
      oldcrc, so we can use it across multiple blocks.
*/

crc(oldcrc,blk,sz)

unsigned oldcrc;
char  *blk;
long sz;

{
      register unsigned crcret;
      register char *ptr;
      register long cnt;
      register unsigned i;

      crcret = oldcrc; ptr = blk; cnt = sz;

      while (cnt--) {
            crcret = crcret ^ ( ((int) *ptr++) << 8);
            for (i = 0; i < 8; i++)
                  if (crcret & 0x8000) crcret = (crcret<<1) ^
0x1021;
                  else crcret <<= 1;
      };

      return(crcret);
}
```

This algorithm is acceptably fast, but it's fairly simple to produce an even faster, table-driven version.

Using a CRC checks that the data survived transport regardless of how many hops were involved and no matter how poor the error detection and correction of any of those hops. I believe this to be especially important in this sort of program since corruption of the contents of the file can lead to executing random data or creating a file with incorrect header information. The consequences can be damaged file systems, confused users, and maybe even lost data.

PackIt and MacBinary

MacBinary is a widely accept standard file format for transmitting single Macintosh files across a telecommunications channel. It's quite possible for the PackIt and MacBinary protocols to live completely separately. In this case, a group of files is packed with PackIt, then transported with MacBinary (or other protocols), then unpacked. This sort of separation has been accepted on other machines, and has thus far been the normal mode of operation for Macintosh PackIt files. It does make a two step process out of file transfer, however, which takes more time and is more difficult for the user to understand.

The disadvantages of combining the unpacking and transfer process are the following: (1) Real-time unpacking makes it difficult for the user to be given a choice during the unpacking process — the current PackIt program suggests a file name but does not require the user to use it. And (2) with a two step process, a number of extensions to PackIt immediately suggest themselves, including data compression, encryption, etc.; this rich a set of transformations will be very difficult, probably impossible to standardize, and therefore impossible to include in each implementation of PackIt-style unpacking.

[Note: If you consider the PackIt protocol as described above to be a new version of MacBinary, there are two ways in which it can be reconciled with the current MacBinary. The first is to add a MacBinary version byte at the start of the PackIt protocol. The other approach is to use the initial 'P' in the 'PMag' magic number as the version. In other words, PackIt would be MacBinary version 80.]

Future Extensions

Because of the sub-file "magic number," it is possible for multiple sub-file formats to coexist in PackIt, even within the same packed file. This allows a number of possibilities for the future, including:

- Data compression.
- Multiple compression schemes depending upon the type of data being compressed, with the type given in the magic number.
- Data encryption.

It is in everyone's best interest to coordinate any extentions of the file format, and I'm more than happy to act as "keeper of the protocol." Therefore, if anyone wishes to add a new sub-file type to the PackIt protocol "standard," simply let me know what magic number you wish to use (it should start with 'P', but can have pretty much any three letters after that). If you wish to make the protocol public, also send a description of it and I'll add it to this paper as an appendix.

Harry R. Chesley
1850 Union St., #360
San Francisco, CA 94123

Outside Outside Macintosh

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

VideoWorks Object Code for Developers

By Marc Canter, MacroMind, Inc.

VideoWorks (in case you don't already know) is a full-fledged animation program that features 24 sprites on the screen at once. It's also the first such program for the Macintosh. Because of this, we've been bombarded by folks requesting use of our code, so that they can include animation in their programs.

Well, we've finally decided to make our code available, and we're going to support it ourselves! Because of the wide range of applications that people are interested in implementing, we have developed a multi-tier approach toward sublicensing. The amount of royalties will be based on the different levels of code usage.

Quickly summed up, if you'd like to include animation in your "About the..." window, or just plain jazz up your program, then we'll barely charge you anything for the code (just initial costs, a small license fee, and tech support). If you'd like to produce a full-fledged video game — that uses animation throughout, then we'll charge more for licensing (going up to a maximum of 60 cents a unit).

The range of possibilities in unlimited. For instance, comprehensive on-line guided tours are now possible; in fact, the example program we provide does just that. Existing VideoWorks documents can be used in final code, converted into a standard resource, and called up at any time, from anywhere in your program. Animation can add life and spirit to many different types of products: A strategy game could play out animated battles; a trivia game could use animation as a reward.

The interface will consist of a jump table that is accessed at the start of the object code. Standardized functions (in each of various languages) will be included with the package to pass parameters and call the available routines.

Functions include:

- Starting and stopping VideoWorks documents.
- Isolating and repeating animation channels.
- Maintaining the background processes involved in animation.
- Writing images onto the screen in various modes (Xor, Or, Copy, NotOr), and executing QuickDraw primitives.
- Playing back sounds effects and one-line melodies. (We will also be supporting the various sampling boxes that are becoming available.)
- Changing frame rates (tempo) and background screen (block or white).

Sample programs usable by Lisa Pascal, MDS Assembler, C, and so forth, will be available, as will the actual VideoWorks relocatable object code.

Producing *MusicWorks, Art Grabber,* and *VideoWorks* has kept us very busy, and that's why we've waited so long to announce this offer. Most of the costs incurred involve technical documentation and technical support, and that's why we made out licensing fees so low, and out technical support a bit higher. If you are an experienced Macintosh developer, we think you'll need only minimal support. If, however, you call us up a lot, and ask lots of questions, then we'll have to charge you per hour for support.

Obviously, the quality of the technical documentation will affect how often even the most experienced programmer will have to call. We have every intention of producing the best quality documentation possible, to save everybody time and money. (Believe me, we've seen some bad documentation in our days, so we know what to avoid.)

No products will be allowed that directly compete with MacroMind products — no animation creativity tools, guided tours, or other types of slide show products. This doesn't mean that you can't do a guided tour or slide show of your own product, but we won't allow anyone to sell a *tool* to do these functions using our code.

MacroMind is also announcing a "Guided Tour Authoring System" we've developed for users who are not full-fledged programmers. This system provides a quick and easy way to create sophisticated guided tours. It allows you to define sprites as "buttons," so that when the user presses them, the animation will pause, continue, or branch to another document. The resulting collection of VideoWorks documents and "Tour Engine" can be combined into a single, double-clickable application. Licensing arrangements are similar to those for the VideoWorks object code.

For more information, contact Erik Neumann at MacroMind, (312)327-5821.