ong, 4 bytes are being put on the stack, and an integer is only two bytes. Even though one instruction is being used, there are actually two parameters being passed to the MoveTo trap.

```
  50E2: A893           '..'             _MoveTo ; (h,v:INTEGER)
  50E4: 3F3C 0029      '?<.)'           PUSH    #41
  50E8: 4EBA CE84      2001F6E          JSR     DRAWRESS
```

It turns out that DRAWRESS will draw the 41st string in the STR# resource. If you look in Resedit, you will see that this is "3.8" the version number.

```
  50EC: 3F3C 000C      '?<..'           PUSH    #12
```

Note the lack of a size specifier. Remember that this means use the word (two bytes) size. Textsize needs an integer and IM tells us that an integer is two bytes - or one word.

```
  50F0: A88A           '..'             _TextSize ; (size:INTEGER) This is
pretty easy - sets the fontsize to 12 point.
  50F2: 422D F4EF        -$B11          CLR.B   glob25(A5)
```

Here is the .B size specifier, meaning clear only the low byte of glob25.

```
  50F6: 42A7           'B.'             CLR.L   -(A7)
  50F8: 3F3C 0004      '?<..'           PUSH    #4
  50FC: A9B9           '..'             _GetCursor ;
(cursorID:INTEGER):CursHandle
```

OK, this is a slightly different trap, since it returns something on the stack - as evidenced by the colon and description at the end of the trap parameter list (:CursHandle). Since this trap returns a value on the stack (and not with a passed pointer as with the GWMgrPort above), the program will first clear enough stack space to hold that value. Thus the CLR.L -(A7). The trap returns a handle which is 32 bits or a long word. The trap needs an integer, so the program pushes the word 4 onto the stack. Next, the program will pop the CursHandle returned by the trap off the stack into the variable glob24.

```
  50FE: 2B5F F4EA        -$B16          POP.L   glob24(A5)
```

This the CursorHandle.

```
  5102: 1F3C 0002      '.<..'           PUSH.B  #2
  5106: 4EBA AEF8      2000000          JSR     SETTHECU
```

This subroutine is setting the cursor. If you look at it, you will see that it looks at the parameter passed (2 in this case) as well as glob25 (0 in this case). When called from here, it will pass down to the 2nd SetCursor and use the CursorHandle in glob24.

```
  510A: 42A7           'B.'             CLR.L   -(A7)
```

Once again, clear space on the stack for a returned handle.

```
  510C: 2F3A 0144      2005252          PUSH.L  data260     ; 'PACK'
  5110: 3F3C 0003      '?<..'           PUSH    #3
```

GetResource needs the resource type and the ID# to load.

```
  5114: A9A0           '..'             _GetResource ; (theType:ResType;
ID:INTEGER):Handle
  5116: 285F           '(_'             POP.L   A4
```

Pop the handle (to the PACK resource) into A4.

```
  5118: 2F0C           '/.'             PUSH.L  A4
```

And push it back on the stack so HNoPurge can use it.

```
  511A: 4EAD 00CA      1000AA6          JSR     HNoPurge(A5)
```

Once again we see a subroutine with the same name as a trap. You can bet that the trap will be called somewhere in the subroutine.

```
  511E: 42A7           'B.'             CLR.L   -(A7)
  5120: 2F3A 0130      2005252          PUSH.L  data260     ; 'PACK'
  5124: 3F3C 0006      '?<..'           PUSH    #6
  5128: A9A0           '..'             _GetResource ; (theType:ResType;
ID:INTEGER):Handle
  512A: 285F           '(_'             POP.L   A4
  512C: 2F0C           '/.'             PUSH.L  A4
  512E: 4EAD 00CA      1000AA6          JSR     HNoPurge(A5)
```
OK, the previous several lines have basically loaded two resources, PACK #3, and PACK #6. The handles to the two resources have been made non-purgeable meaning that the memory manager will not remove them to create free space.
```
  5132: 42A7           'B.'             CLR.L   -(A7)
  5134: 3F3C 0001      '?<..'           PUSH    #1
  5138: 4EAD 0182      1000D8C          JSR     proc61(A5)
```
This little gem invokes Pack6. My understanding of the package manager is less than it should be, but it looks to me like this says do a Pack6 with a selector of 1. Hell, lets just look at proc 61...

```
   D8C: 7406           't.'    proc61   MOVEQ   #6,D2
```
OK, here is the selector (and not the 1 passed from the above procedure). So we are going to be calling the IUGetIntl procedure (I think) with a parameter of 1 (passed from the calling procedure. Look in IM for details of this trap and its parameters.
```
   D8E: 205F           ' _'             POP.L   A0
```
This pops the parameter passed,
```
   D90: 3F02           '?.'             PUSH    D2
```
so that the selector parameter can be put ahead of it on the stack.
```
   D92: 2F08           '/.'             PUSH.L  A0
```
Now the 2nd parm can be put back on the stack and the trap called.
```
   D94: ADED           '..'             _Pack6  AutoPop; (selector:INTEGER)
```

```
  513C: 285F           '(_'             POP.L   A4
```
proc 61 is returning a handle to the intl resource that it loaded, so save it in A4.
```
  513E: 2F0C           '/.'             PUSH.L  A4
  5140: 4EAD 00CA      1000AA6          JSR     HNoPurge(A5)
  5144: 42A7           'B.'             CLR.L   -(A7)
  5146: 2F3A 010A      2005252          PUSH.L  data260     ; 'PACK'
  514A: 3F3C 0007      '?<..'           PUSH    #7
  514E: A9A0           '..'             _GetResource ; (theType:ResType;
ID:INTEGER):Handle
  5150: 285F           '(_'             POP.L   A4
```
A4 now has a handle to Pack #7.
```
  5152: 2F0C           '/.'             PUSH.L  A4
  5154: 4EAD 00CA      1000AA6          JSR     HNoPurge(A5)
  5158: 4EAD 0172      1000D7C          JSR     proc59(A5)
```
This proc calles Pack2 with a selector of 2. This reads the Disk Initialization package into memory.
```
  515C: 42A7           'B.'             CLR.L   -(A7)
```
Clear space on stack for a returned handle.
```
  515E: 2F3A 00EE      200524E          PUSH.L  data259     ; 'ICON'
  5162: 4267           'Bg'             CLR     -(A7)
```
Push the integer 0.
```
  5164: A9A0           '..'             _GetResource ; (theType:ResType;
ID:INTEGER):Handle
  5166: 285F           '(_'             POP.L   A4
```
A4 has a handle to Icon resource ID 0.
```
  5168: 42A7           'B.'             CLR.L   -(A7)
  516A: 2F3A 00E2      200524E          PUSH.L  data259     ; 'ICON'
  516E: 3F3C 0001      '?<..'           PUSH    #1
  5172: A9A0          '..'             _GetResource ; (theType:ResType;
ID:INTEGER):Handle
  5174: 285F          '(_'             POP.L   A4
```
A4 has a handle to Icon resource ID 1.
```
  5176: 4267          'Bg'             CLR     -(A7)
```
Make space for the returned RefNum.
```
  5178: A994          '..'             _CurResFile ; :RefNum
```
Note - no parameters passed.
```
  517A: 3B5F FFE0        -$20          POP     glob58(A5)
```
Pop off the returned RefNum.
```
  517E: 486D FEDE       -$122          PEA     glob56(A5)
  5182: 3F3C 000D     '?<..'           PUSH    #13
  5186: 4EAD 002A     100048C          JSR     proc5(A5)
```
Here is proc5 again - the string getter. If you remember (from looking at DRAWRESS), the 1st parm is the string ptr, and the 2nd is the string # to get. This is returning a ptr to the string "The quick brown fox..." in glob56.
```
  518A: 7000          'p.'             MOVEQ   #0,D0
  518C: 2B40 FED4       -$12C          MOVE.L  D0,glob52(A5)
  5190: 7000          'p.'             MOVEQ   #0,D0
  5192: 2B40 FECC       -$134          MOVE.L  D0,glob50(A5)
  5196: 7000          'p.'             MOVEQ   #0,D0
  5198: 2B40 F61E       -$9E2          MOVE.L  D0,glob41(A5)
  519C: 3B7C FFFF F616  -$9EA          MOVE    #$FFFF,glob38(A5)
  51A2: 426D F614       -$9EC          CLR     glob37(A5)
  51A6: 7000          'p.'             MOVEQ   #0,D0
  51A8: 2B40 F610       -$9F0          MOVE.L  D0,glob36(A5)
  51AC: 7000          'p.'             MOVEQ   #0,D0
  51AE: 2B40 F61A       -$9E6          MOVE.L  D0,glob40(A5)
  51B2: 7034          'p4'             MOVEQ   #52,D0
  51B4: 2B40 F5FE       -$A02          MOVE.L  D0,glob31(A5)
```
The above instructions have simply initialized several global variables. We don't care what they mean at this point. If you like, you can write down what has been set to what, but I would only recommend this if later on you need to know explicitly what a global contains.
```
  51B8: 42A7          'B.'             CLR.L   -(A7)
  51BA: 7002          'p.'             MOVEQ   #2,D0
```
Note the MoveQ. Remember, this is the same as MOVE.L (except it executes faster).
```
  51BC: 2F00          '/.'             PUSH.L  D0
  51BE: 4EAD 009A     1000A5C          JSR     NewHandle(A5)
```
NewHandle is a trap that returns a handle to a block of memory whose size is in D0. It makes sense to guess that this procedure will do essentially the same thing - and after checking, it certainly does.
```
  51C2: 2B5F F622       -$9DE          POP.L   glob42(A5)
```
So glob42 has a handle to a 2 byte chunk of memory.
```
  51C6: 426D F626       -$9DA          CLR     glob43(A5)
  51CA: 70FF          'p.'             MOVEQ   #-1,D0
```
Here is one of those cases where the sign bit is important. Remember that the -1 is sign extended to 32 bits so D0 is being set to all binary ones (-1 in binary).
```
  51CC: 2B40 F602       -$9FE          MOVE.L  D0,glob32(A5)
  51D0: 42A7          'B.'             CLR.L   -(A7)
  51D2: 2EB8 02F0         $2F0         MOVE.L  DoubleTime,(A7)
  51D6: 7002          'p.'             MOVEQ   #2,D0
  51D8: 2F00          '/.'             PUSH.L  D0
  51DA: 4EAD 01A2     1001120          JSR     proc76(A5)
```
This is a gross looking (i.e. no Traps anywhere) procedure so I am not going

to attempt to figure it out. You will want to use the technique a lot (the "Too Gross" technique) to determine which procedures to spend time with.

```
  51DE: 2B5F F5F6        -$A0A          POP.L   glob29(A5)
  51E2: 207C 0000 0AD8    $AD8          MOVEA.L #SysResName,A0
```

Put a pointer to the System File's name in A0.

```
  51E8: 43ED F4F6        -$B0A          LEA     glob28(A5),A1
```

Put the address of glob28 in A1.

```
  51EC: 703F           'p?'             MOVEQ   #63,D0
```

Set up D0 as a loop counter.

```
  51EE: 22D8           '".'    lho_3    MOVE.L  (A0)+,(A1)+
```

This moves 4 bytes from A0 to A1. Note the use of auto post increment to automatically move the pointers to the next available data each time. This moves 4 bytes of the System name into glob28. Note that glob28 will not be a pointer to the Sys Name, but will rather contain the actual string data.

```
  51F0: 51C8 FFFC      20051EE          DBRA    D0,lho_3
```

This decrements D0 (the loop counter) and branches back to the start of the loop until it is finished.

```
  51F4: 422D F4F5        -$B0B          CLR.B   glob27(A5)
  51F8: 267C 0000 028E    $28E          MOVEA.L #Rom85,A3
```

ROM85 is another of those variables that my old IMs are missing so god only knows what is going on here. I'll guess that it is looking for the 128K roms.

```
  51FE: 4A53           'JS'             TST     (A3)
  5200: 6D20           2005222          BLT.S   lho_4
  5202: 42A7           'B.'             CLR.L   -(A7)
  5204: 3F3C 008F      '?<..'           PUSH    #143
  5208: 4EAD 00E2      1000AC6          JSR     proc38(A5)
```

Well, let's see here. Proc38 uses the passed parm as a trap number and returns that traps address on the stack.

```
  520C: 42A7           'B.'             CLR.L   -(A7)
```

Note that the trap address has not been popped off the stack. So when these next instructions are done, that address will still be on the stack.

```
  520E: 3F3C 009F      '?<..'           PUSH    #159
  5212: 4EAD 00E2      1000AC6          JSR     proc38(A5)
```

Get another trap address on the stack,

```
  5216: 201F           ' .'             POP.L   D0
```

and put it in D0, leaving the first trap address on the stack.

```
  5218: B09F           '..'             CMP.L   (A7)+,D0
```

Now, compare the two trap addresses,

```
  521A: 56C0           'V.'             SNE     D0
```

and set the low byte of D0 to FF hex if they are not the same.

```
  521C: 4400           'D.'             NEG.B   D0
```

Do 2's complement - make the low byte of D0 its own negative. Since D0's byte is either 0 or FF (from the SNE), the NEG will make it either 0 (if it was 0) or 1 (if it was FF) - (for NEG, invert the bits, then add a binary 1).

```
  521E: 1B40 F4F5        -$B0B          MOVE.B  D0,glob27(A5)
```

And save this number.

```
  5222: 42A7           'B.'    lho_4    CLR.L   -(A7)
  5224: 2F3C 0001 0000 '/<....'         PUSH.L  #$10000
  522A: 4EAD 009A      1000A5C          JSR     NewHandle(A5)
```

Get a new Handle for a block of size 10000 hex.

```
  522E: 2B5F F4F0        -$B10          POP.L   glob26(A5)
```

And save the handle.

```
  5232: 6708           200523C          BEQ.S   lho_5
```

Branch if a NIL pointer (meaning the memory was not available) is popped off the stack.

```
  5234: 487A FE26      200505C          PEA     MYGROWZO
```

Otherwise setup a grow zone function.

```
  5238: 4EAD 0092      1000A1E          JSR     SetGrowZone(A5)
```
A grow zone procedure is a custom method for handling low memory conditions and overrides the memory managers routines. Not a great description, but we don't really care about this.
```
  523C: 4CDF 1880      'L...'  lho_5    MOVEM.L (A7)+,D7/A3-A4
```
Restore those saved regs,
```
  5240: 4E5E           'N^'             UNLK    A6
```
Kill the stack frame,
```
  5242: 4E75           'Nu'             RTS
```
And return to the caling proc.

```
  5244: D345 5455 5020 2020    data257  DNAME   SETUP   ,0


  524C: '..'                   data258  DC.W    8

                               ;-refs -  2/SETUP

  524E: 4943                   data259  DC.B    'ICON'

                               ;-refs -  2/SETUP

  5252: 5041                   data260  DC.B    'PACK'
```

The DRAWRESS Procedure
```
  1F6E:                                 QUAL    DRAWRESS ; b# =284  s#2
=proc148

                               vfp_1     VEQU  -256
```
One local variable.
```
                               param1    VEQU  8
```
One parameter needed.
```
  1F6E:                                 VEND

                               ;-refs -  2/DRAWFHIN   2/SETUP      2/DRAWNUM

                               ;-        2/DRAWDHIN
```
OK, you should be able to just look at this and see what happens. First off, look at the trap, DrawString. It takes one parameter, a pointer to a string. Now, the previous line says to push the address of the local variable so this has to be the string pointer. Go back a few lines and we see that proc5 is being called with two parameters: the string pointer, and the parameter from the calling procedure. You can deduce that proc5 has to get a string from somewhere, and probably will call the GetString trap or some equivalent. In fact, if you look at proc5, you will see that it calls GetResource (resource type STR#). This returns a handle to the STR# resource. Proc5 then uses the second parameter to figure out which string the calling procedure really wants. Proc5 loops through the STR# resource until it comes to the right string, then moves a pointer to the string into the first parameter and returns. When it gets back here, vfp_1 contains a pointer to the string.
```
  1F6E: 4E56 FF00      'NV..'  DRAWRESS LINK    A6,#-$100
  1F72: 486E FF00      200FF00          PEA     vfp_1(A6)
  1F76: 3F2E 0008      2000008          PUSH    param1(A6)
  1F7A: 4EAD 002A      100048C          JSR     proc5(A5)
  1F7E: 486E FF00      200FF00          PEA     vfp_1(A6)
```
At this point, vfp_1 has the stringptr.
```
  1F82: A884           '..'             _DrawString ; (s:Str255)
  1F84: 4E5E           'N^'             UNLK    A6
  1F86: 205F           ' _'             POP.L   A0
  1F88: 544F           'TO'             ADDQ    #2,A7
  1F8A: 4ED0           'N.'             JMP     (A0)
```
Note that there is no RTS instruction to return. The subroutine uses a common substitute. First it pops the return address off the stack (which is actually what the RTS would have done anyways) and then does an indirect JMP (A0). This just means to jump to whatever A0 points to and A0 points to the return address.

```
  1F8C: C452 4157 5245 5353    data125  DNAME   DRAWRESS,0,0
```

The MAKEAWIN Procedure

```
  5852:                                 QUAL    MAKEAWIN ; b# =490  s#2
=proc209


                               vhy_1     VEQU  -12
```
Two local variables, no parms passed.
```
                               vhy_2     VEQU  -8
  5852:                                 VEND

                               ;-refs -  1/DA Mover

  5852: 4E56 FFF0      'NV..'  MAKEAWIN LINK    A6,#-$10
  5856: 42A7           'B.'             CLR.L   -(A7)
```
These instructions are setting up the GetNewDialog below. 1st, clear space for the DialogPtr.
```
  5858: 3F3C 000A      '?<..'           PUSH    #10
```
Push the Dialog ID #.
```
  585C: 42A7           'B.'             CLR.L   -(A7)
```
Push a NIL pointer for wStorage
```
  585E: 70FF           'p.'             MOVEQ   #-1,D0
  5860: 2F00           '/.'             PUSH.L  D0
```
Push a 32 bit -1 (IM says to do this to make the dialog the frontmost window).
```
  5862: A97C           '.|'             _GetNewDialog ; (DlgID:INTEGER;
wStorage:Ptr; behind:WindowPtr):DialogPtr
  5864: 2B5F FFFA           -6          POP.L   glob67(A5)
```
And pop off the dialogPtr. This will be used by proc MAKEBOX.
```
  5868: 486D FEC4        -$13C          PEA     glob48(A5)
  586C: 3F3C 000A      '?<..'           PUSH    #10
```
This is the dialog item - the left list box if you check Resedit.
```
  5870: 4EBA FF32      20057A4          JSR     MAKEBOX
```
Well, after inspecting this procedure, it looks like more can be determined by just looking at these few instructions here. Notice that MakeBox is being called with two parameters: The 1st being an unknown global variable, and the second being one of the two list boxes in Mover's main dialog. So it looks like MakeBox is just performing some housekeeping on these two list boxes.
```
  5874: 486D FEC8        -$138          PEA     glob49(A5)
  5878: 3F3C 000B      '?<..'           PUSH    #11
```
Now do the right list box.
```
  587C: 4EBA FF26      20057A4          JSR     MAKEBOX
  5880: 206D FEC4        -$13C          MOVEA.L glob48(A5),A0
```
Get the address in (not of) glob48 into A0,
```
  5884: 2050           ' P'             MOVEA.L (A0),A0
```
and dereference it - or get whatever glob48 was pointing at into A0.
```
  5886: 216D FEC8 0004   -$138          MOVE.L  glob49(A5),4(A0)
```
Now move glob49 (a pointer I suspect) into 4 past A0. So glob48 contains a pointer which points four bytes behind the pointer in glob49.
```
  588C: 206D FEC8        -$138          MOVEA.L glob49(A5),A0
```
Now do the exact opposite. Grab the pointer in glob49 and stick the pointer in glob48 4 bytes past it.
```
  5890: 2050           ' P'             MOVEA.L (A0),A0
  5892: 216D FEC4 0004   -$13C          MOVE.L  glob48(A5),4(A0)
```

These last few instructions were kind of a mess because we don't no anything about how globs 48 and 49 will be used. We will come back here after looking at MainEven and particularly HandleBu. It will turn out that these two globals are pointers (or maybe handles, we don't really care) to the two list boxes on the main dialog. In addition, each pointer as a way of referring to the other list box. At this point, this does not make any sense, but later on, glob 50 will be set to either glob48 or glob 49 (or NIL) depending on which list box - if any - has a selection made in it. The reason that glob48 and glob49 need to refer to each other, is that glob50 will be used to check both list boxes to see if their associated volumes are locked. See HandleBu for details.
```
  5898: 2F2D FFFA           -6          PUSH.L  glob67(A5)
  589C: 3F3C 0002      '?<..'           PUSH    #2
```
Item is the Copy button.
```
  58A0: 486E FFF4      200FFF4          PEA     vhy_1(A6)
  58A4: 486D FFF6           -$A         PEA     glob66(A5)
```
This will save a handle to it.
```
  58A8: 486E FFF8      200FFF8          PEA     vhy_2(A6)
  58AC: A98D           '..'             _GetDItem ; (dlg:DialogPtr;
itemNo:INTEGER; VAR kind:INTEGER; VAR item:Handle; VAR box:Rect)
  58AE: 2F2D FFFA           -6          PUSH.L  glob67(A5)
  58B2: 3F3C 0006      '?<..'           PUSH    #6
```
Item is the left Open button.
```
  58B6: 486E FFF4      200FFF4          PEA     vhy_1(A6)
  58BA: 486D FFEC          -$14         PEA     glob63(A5)
```
This will save a handle to it.
```
  58BE: 486E FFF8      200FFF8          PEA     vhy_2(A6)
  58C2: A98D           '..'             _GetDItem ; (dlg:DialogPtr;
itemNo:INTEGER; VAR kind:INTEGER; VAR item:Handle; VAR box:Rect)
  58C4: 2F2D FFFA           -6          PUSH.L  glob67(A5)
  58C8: 3F3C 0007      '?<..'           PUSH    #7
```
Item is the right Open button.
```
  58CC: 486E FFF4      200FFF4          PEA     vhy_1(A6)
  58D0: 486D FFF0          -$10         PEA     glob64(A5)
```
This will save a handle to it.
```
  58D4: 486E FFF8      200FFF8          PEA     vhy_2(A6)
  58D8: A98D           '..'             _GetDItem ; (dlg:DialogPtr;
itemNo:INTEGER; VAR kind:INTEGER; VAR item:Handle; VAR box:Rect)
```
Now the program is going to assign dialog procedures to various of its items. Items 12 and 13 - the two filename boxes are assigned the DrawName proecdure.

Items 14 - the size selected box - gets DrawSize. Item 15 -the font text demo box - gets DrawHint. Items 16 through 18 - various lines in the dialog box - get DrawGray. And items 19 and 20 - the free space on disk boxes - get DrawFree. If you examine SetDProc, you will see that it simply invokes GetDItem to get a handle to the dialog item (passed from the list below) and then uses SetDItem to set the dialogProcPtr to the procedure passed from the list below.

```
  58DA: 3F3C 000C      '?<..'           PUSH    #12
  58DE: 487A FB2E      200540E          PEA     DRAWNAME
  58E2: 4EBA FE7E      2005762          JSR     SETDPROC
  58E6: 3F3C 000D      '?<..'           PUSH    #13
  58EA: 487A FB22      200540E          PEA     DRAWNAME
  58EE: 4EBA FE72      2005762          JSR     SETDPROC
  58F2: 3F3C 000E      '?<..'           PUSH    #14
  58F6: 487A FC32      200552A          PEA     DRAWSIZE
  58FA: 4EBA FE66      2005762          JSR     SETDPROC
  58FE: 3F3C 000F      '?<..'           PUSH    #15
  5902: 487A FA3A      200533E          PEA     DRAWHINT
  5906: 4EBA FE5A      2005762          JSR     SETDPROC
  590A: 3F3C 0010      '?<..'           PUSH    #16
  590E: 487A FE1C      200572C          PEA     DRAWGRAY
  5912: 4EBA FE4E      2005762          JSR     SETDPROC
  5916: 3F3C 0011      '?<..'           PUSH    #17
  591A: 487A FE10      200572C          PEA     DRAWGRAY
  591E: 4EBA FE42      2005762          JSR     SETDPROC
  5922: 3F3C 0012      '?<..'           PUSH    #18
  5926: 487A FE04      200572C          PEA     DRAWGRAY
  592A: 4EBA FE36      2005762          JSR     SETDPROC
  592E: 3F3C 0013      '?<..'           PUSH    #19
  5932: 487A FD12      2005646          PEA     DRAWFREE
  5936: 4EBA FE2A      2005762          JSR     SETDPROC
  593A: 3F3C 0014      '?<..'           PUSH    #20
  593E: 487A FD06      2005646          PEA     DRAWFREE
  5942: 4EBA FE1E      2005762          JSR     SETDPROC
  5946: 2F2D FFFA           -6          PUSH.L  glob67(A5)
Now the dialog is made the current Port
  594A: A873           '.s'             _SetPort ; (port:GrafPtr)
  594C: 2F2D FFFA           -6          PUSH.L  glob67(A5)
and make the dialog visible,
  5950: A915           '..'             _ShowWindow ; (theWindow:WindowPtr)
  5952: 2F2D FFFA           -6          PUSH.L  glob67(A5)
and make it the frontmost window.
  5956: A91F           '..'             _SelectWindow ; (theWindow:WindowPtr)
  5958: 3F3C 0002      '?<..'           PUSH    #2
  595C: 4EBA A78A      20000E8          JSR     DIMITEM
These instructions dim the two Open buttons.
  5960: 3F3C 0003      '?<..'           PUSH    #3
  5964: 4EBA A782      20000E8          JSR     DIMITEM
  5968: 2F2D FFFA           -6          PUSH.L  glob67(A5)
  596C: A981           '..'             _DrawDialog ; (dlg:DialogPtr) And
finally, draw the damn thing.
  596E: 4E5E           'N^'             UNLK    A6
  5970: 4E75           'Nu'             RTS


  5972: CD41 4B45 4157 494E    data270  DNAME   MAKEAWIN,0,0
```

```
The MAKEBOX Procedure.
  57A4:                             QUAL    MAKEBOX ; b# =488  s#2
=proc208

                              vhx_1     VEQU  -14
                              vhx_2     VEQU  -10
                              vhx_3     VEQU  -8
                              vhx_4     VEQU  -4
                              param2    VEQU  8
Parm 2 is the dialog item #
                              param1    VEQU  10
  57A4:                             VEND

                              ;-refs -  2/MAKEAWIN

  57A4: 4E56 FFF2      'NV..'  MAKEBOX  LINK    A6,#-$E
  57A8: 48E7 0018      'H...'           MOVEM.L A3-A4,-(A7)
  57AC: 266E 000A      200000A          MOVEA.L param1(A6),A3
A3 gets whatever is in parm 1.
  57B0: 2F2D FFFA           -6          PUSH.L  glob67(A5)
Push the DialogPtr,
  57B4: 3F2E 0008      2000008          PUSH    param2(A6)
And push the item #.
  57B8: 486E FFF6      200FFF6          PEA     vhx_2(A6)
This will get the Kind.
  57BC: 486E FFF2      200FFF2          PEA     vhx_1(A6)
This will get the ItemHandle.
  57C0: 486E FFF8      200FFF8          PEA     vhx_3(A6)
This will get the Box.
  57C4: A98D           '..'             _GetDItem ; (dlg:DialogPtr;
itemNo:INTEGER; VAR kind:INTEGER; VAR item:Handle; VAR box:Rect)
  57C6: 2F2D FFFA           -6          PUSH.L  glob67(A5)
Now push the dialogPtr and item again...
  57CA: 3F2E 0008      2000008          PUSH    param2(A6)
  57CE: 3F2E FFF6      200FFF6          PUSH    vhx_2(A6)
Push the item Kind
  57D2: 487A F662      2004E36          PEA     DRAWBOX
See IM - this is a procPtr.
  57D6: 486E FFF8      200FFF8          PEA     vhx_3(A6)
And push the Box
  57DA: A98E           '..'             _SetDItem ; (dlg:DialogPtr;
itemNo,kind:INTEGER; item:Handle; box:Rect)
  57DC: 42A7           'B.'             CLR.L   -(A7)
  57DE: 7064           'pd'             MOVEQ   #100,D0
  57E0: 2F00           '/.'             PUSH.L  D0
  57E2: 4EAD 009A      1000A5C          JSR     NewHandle(A5)
  57E6: 269F           '&.'             POP.L   (A3)
Get a new handle - size 100 - and put it into parm1 (which A3 points to).
  57E8: 2053           ' S'             MOVEA.L (A3),A0
A0 gets the handle.
  57EA: 2850           '(P'             MOVEA.L (A0),A4
And A4 gets the pointer. OK, A0 is a handle meaning it points to a pointer
which in turn points to whatever it is we care about (in this case, a free
block of memory).  That means that (A0) grabs what ever A0 points to which is
```

(by definition of a handle) the pointer.

```
  57EC: 28AD FFFA           -6          MOVE.L  glob67(A5),(A4)
```

And now we put the dialogPtr into the block of memory gotten by NewHandle.

```
  57F0: 426C 0060      'Bl.`'           CLR     96(A4)
```

Remember, A4 points (its a pointer, not a handle!) to a block of memory, 100 bytes long. So this instruction simply clears the 96 byte in that block.

```
  57F4: 204C           ' L'             MOVEA.L A4,A0
```

Put the pointer into A0.

```
  57F6: 5088           'P.'             ADDQ.L  #8,A0
```

Add 8 to A0. Previously we had stored the dialogPtr at the beginning of this block. Since a pointer is 8 bytes long, A0 no points to the first byte after the dialogPtr.

```
  57F8: 43EE FFF8      200FFF8          LEA     vhx_3(A6),A1
```

vhx_3 is a Box which is of type Rect which is 4 integers, or 4 words, or two long words.

```
  57FC: 20D9           ' .'             MOVE.L  (A1)+,(A0)+
  57FE: 20D9           ' .'             MOVE.L  (A1)+,(A0)+
```

So move the Box information into the free memory right after the dialogPtr and increment A0 to the next free byte.

```
  5800: 302E FFFC      200FFFC          MOVE    vhx_4(A6),D0
```

This is tough since we don't know what vhx_4 is to start with.

```
  5804: 906E FFF8      200FFF8          SUB     vhx_3(A6),D0
```

But whatever, subtrack vhx_3 from it, result in D0.

```
  5808: 48C0           'H.'             EXT.L   D0
```

At this point, D0 is accurate to the word length (since that was all the SUB specified). This will make it's sign (negative or posative) accurate to all 32 bits.

```
  580A: 81FC 0010      '....'           DIVS    #16,D0
```

Now, divide by 16.

```
  580E: 3940 0062      '9@.b'           MOVE    D0,98(A4)
```

And put this value (whatever it is) in the last two bytes (notice it is a word length instruction) of the memory block.

```
  5812: 426C 0058      'Bl.X'           CLR     88(A4)
  5816: 397C FFFF 0056 '9|...V'         MOVE    #$FFFF,86(A4)
  581C: 422C 0014      'B,..'           CLR.B   20(A4)
```

These last instructions are filling in various parts of the memory block.

```
  5820: 206D FFFA           -6          MOVEA.L glob67(A5),A0
```

Put the DialogPtr back in A0.

```
  5824: 2153 0098      '!S..'           MOVE.L  (A3),152(A0)
```

A3 still points to parm1.

```
  5828: 2F13           '/.'             PUSH.L  (A3)
```

So, this effectively pushes parm1

```
  582A: 4EBA AEA4      20006D0          JSR     MAKESBAR
```

This is fairly complicated, but this procedure makes a scroll bar for the dialog item.

```
  582E: 2053           ' S'             MOVEA.L (A3),A0
  5830: 2050           ' P'             MOVEA.L (A0),A0
```

Can't tell what these instructions are doing.

```
  5832: 2068 0010      ' h..'           MOVEA.L 16(A0),A0
  5836: 2050           ' P'             MOVEA.L (A0),A0
  5838: 2153 0024      '!S.$'           MOVE.L  (A3),36(A0)
  583C: 4CDF 1800      'L...'           MOVEM.L (A7)+,A3-A4
  5840: 4E5E           'N^'             UNLK    A6
  5842: 205F           ' _'             POP.L   A0
```

Pop off the return address.

```
  5844: 5C4F           '\O'             ADDQ    #6,A7
  5846: 4ED0           'N.'             JMP     (A0)
```

And jump back to the calling procedure.

```
  5848: CD41 4B45 424F 5820    data269  DNAME   MAKEBOX ,0,0
```

The MAINEVEN Procedure

Basically, the main loop consists of a set of housekeeping routines, a call to ModalDialog to read dialog events that take place, and a simple jump table to handle the various events. D7 needs to be zero for the loop to keep running. If an error occurs, or the user hits Quit, D7 is changed to one and the procedure exits. First, DA Mover attempts to allocate a large block of memory (10000 hex) into glob26. If this is successful (or glob26 already has a memory handle) then the program skips down to make some more checks - otherwise a memory error is generated. Next, the procedure checks to see if there are any files open and if so, calls FlushVol to write any changes to disk.

```
     0:                              QUAL    MAINEVEN ; b# =1  s#1  =proc1

                               vab_1     VEQU  -6
     0:                              VEND

                               ;-refs -  1/DA Mover

     0: 4E56 FFF8      'NV..'  MAINEVEN LINK    A6,#-8
     4: 48E7 0308      'H...'           MOVEM.L D6-D7/A4,-(A7)
     8: 4207           'B.'             CLR.B   D7
```

Enable the Main Event Loop.

```
     A: 4AAD F4F0        -$B10 lab_1    TST.L   glob26(A5)
```

glob46 will (or does) contain a handle to a large block of memory. So, if glob26 already has the handle, branch down, otherwise try to get some memory.

```
     E: 661C           100002C          BNE.S   lab_2
    10: 42A7           'B.'             CLR.L   -(A7)
```

Clear stack space for the returned handle.

```
    12: 2F3C 0001 0000 '/<....'         PUSH.L  #$10000
```

Size of memory block needed.

```
    18: 4EBA 0A42      1000A5C          JSR     NewHandle
    1C: 2B5F F4F0        -$B10          POP.L   glob26(A5)
```

And get the handle in glob26.

```
    20: 660A           100002C          BNE.S   lab_2
```

Remember, a NIL handle or pointer is all zeroes. glob26 either has a valid handle or a NIL handle. If it is valid, branch.

```
    22: 3F3C 0032      '?<.2'           PUSH    #50
    26: 4EAD 01CA      200023C          JSR     DOALERT(A5)
```

Otherwise do some memory alert (you can check this if you like.)

```
    2A: 7E01           '~.'             MOVEQ   #1,D7
```

and disable the main event loop.

```
    2C: 1007           '..'    lab_2    MOVE.B  D7,D0
    2E: 6600 00D0      1000100          BNE     lab_15
```

Go if loop disabled from above.

```
    32: 206D FEC4        -$13C          MOVEA.L glob48(A5),A0
```

Get reference to left list box.

```
    36: 2850           '(P'             MOVEA.L (A0),A4
    38: 4A6C 0058      'Jl.X'           TST     88(A4)
```

Look at the descrpition of FlushVol (next paragraph) to see what this variable means.

```
    3C: 670E           100004C          BEQ.S   lab_3
```

Seeing that 88(A4) is the VRefNum, then branch if it is zero (no volume available - i.e. the list box has no opened file in it).

```
    3E: 4267           'Bg'             CLR     -(A7)
```

```
Space for function result (OSErr).
    40: 42A7           'B.'             CLR.L   -(A7)
iovNameP
```