NOTE: This Technical Note has been retired. Please see the Technical Notes page for current documentation.

Technical Note DV520

Device Management Overview Q&As

CONTENTS

This Technical Note contains a collection of archived Q&As relating to a specific topic--questions sent the Developer Support Center (DSC) along with answers from the DSC engineers. Current Q&As can be found on the Macintosh Technical Q&As web site.

[Oct 01 1990]

Extending Macintosh mouse location beyond screen boundary

Date Written: 2/28/92

Last reviewed: 2/28/92

We would like to watch mouse movement directly in our program. This means that we would like to make the mouse disappear and then watch the mouse as it moves left and right. We do not want to have any limits on the movement from the size of the screen or anything like that. For instance, if the user keeps moving to the right, we would like our "x" coordinate tracking variable to keep increasing...

Is there any way to do this easily? I thought of taking out the mouse handler interrupt, watching it there, and then increasing a counter, etc. Is there an easier way?

Below you'll find a generic code sample that shows how to track changes in the mouse position. There are many ways of doing this, but the one included below is pretty straightforward and can be done at the application level with no patching involved.

You started by saying that the cursor would be hidden when you do this. That's good since the method below would look very bad if the cursor were visible. This method is also polling for mouse changes; you could install a VBL task that performs this operation as well, though it probably won't be necessary.

The low-memory globals MTemp and RawMouse contain the current mouse location; they should always equal each other when you reset CrsrNew. CrsrNew simply tells the system that the mouse has changed location. You should be able to drop this routine into any running program. (If you have MPW, modify SAMPLE.P to call FussMouse right after initialize and it works great.)

```
procedure FussMouse;

type    PointPtr=^Point;

var RawMouse:PointPtr;
    MTemp:PointPtr;
    RandPt:Point;
    CrsrNew:ptr;
    CrsrCouple:ptr;
    fred:Longint;
    curH,curV:Longint;
    numStr:str255;
    aWindow:WindowPtr;

begin
    aWIndow:=FrontWindow;
    setPort(aWIndow);
    EraseRect(aWindow^.portRect);
    HideCursor;
{ Environment set up to draw the current location }
    RawMouse:=PointPtr($82C);
    MTemp:=PointPtr($828);
    CrsrNew:=ptr($8CE);
    CrsrCouple:=ptr($8CF);
{ Pointers to the low memory globals are inited...
  RawMouse and MTemp both contain the current absolute mouse position
  CrsrNew is a flag that tells the system when the mouse has changed
  CrsrCouple is what CrsrNew should be set to when the mouse has changed }

    curH:=0;
    curV:=0;
    RawMouse^:=Point($00640064);
    MTemp^:=Point($00640064);
{Base all movement detection off of 100,100 so that we have enough room to detect
all mouse movements in the negative and positive directions.}

    CrsrNew^:=CrsrCouple^;
    repeat
        If $00640064<>Longint(MTemp^) then {Mouse Moved?}
            begin
                RandPt:=MTemp^;  {Get the current mouse location}
                   RawMouse^:=Point($00640064); {Reset Low memory for the
                                                 next time}
                   MTemp^:=Point($00640064);
                   CrsrNew^:=CrsrCouple^;
                   CurH:=curH+(RandPt.h-$0064); {Compute how much the mouse
                                                 has moved}
                   CurV:=CurV+(RandPt.V-$0064); {and add it to our logical
                                                 position}
                   Moveto(10,15);  {now simply report the new position to
                                    the user}
                   EraseRect(aWIndow^.PortRect);
                   NumToString(CurH,NumStr);
                   DrawString(concat(NumStr,','));
                   NumToString(CurV,NumStr);
                   DrawString(NumStr);
            end;
    until Button;
end;
```

System 7.0, Disk First Aid 7.0, and Hard Disk Setup 2.0.4

Date Written: 7/30/91

Last reviewed: 8/1/91

We were told that it would be highly advisable to do a file-by-file back-up of all hard drives, a fresh format, and a restore because of a bug in the B-Tree algorithms that's been fixed in the System 7.0 Golden Master. Can we do this and then return to System 6.0.7 safely? Is Apple going to make any sort of general announcement about this problem? Also, can we expect that it is not a "contagious" problem in any way--that is, can we back things up and restore without worrying about it moving from disk to disk?

———

Apple is recommending that users run the final version of Disk First Aid (version 7.0) on all their hard drives to determine if there are any problems with the B-Trees. The reason for this is that a bug has existed in the Macintosh (HFS) operating system from its inception such that the B-Tree structures for devices would become corrupted under certain EXTREMELY RARE circumstances. If and only if DFA reports a problem that it cannot fix (it will tell you if it can or can't) should users perform a file-by-file back up of their data, then reformat all Apple hard drives using HD Setup before copying files back. Making a mirror backup of your data will only put the damaged B-Tree catalog back onto the drive when you restore, so be sure to make your backup file-by-file.

Apple does not intend to make any general published announcement about this because it is addressed in the System 7.0 documentation. As for the commonly asked question, "Why should we back up our data before upgrading to System 7.0? Are we in danger of losing our data?": Backing up your data on a regular basis is always a good idea, and we are recommending it prior to installing System 7.0 only because there is always a chance of data loss when major changes are made to any system configuration. Since System 7.0 is very different in many ways from previous system software versions, some software packages are no longer compatible under this release and conflicts can potentially cause loss of data integrity.

There has been some confusion as to what, exactly, Disk First Aid version 7.0 can do. This version can detect problems with the B-Tree structures (whereas previous versions could not--the only clue it would offer is the "unable to verify status of disk" error). It cannot fix the damage, however. Another bug that DFA 7.0 can correct lies in the Directory Valance. This bug went generally unnoticed for years; it usually resulted in VERY INfrequent occurrences of some annoying behaviors such as not being able to throw away an empty folder (remember the old "Trash could not be emptied because a file was busy or locked" error?). Like the B-Tree structure corruption, this is a bug, not a virus, and is in no way contagious.

If you reformat the drive then reinstall System 6.0.x, you run the risk of damaging the B-Tree structures again because the bug is still present in the B-Tree Manager in ROM. Again, however, let me reiterate that it isn't easy to take it that far. Suffice it to say, you will, in fact, be almost assuredly safe if you run DFA 7.0 occasionally to make sure your B-Tree structures are OK. If it says everything is fine, it is. If not, back up your data and reformat.

Another important step in your upgrade to System 7.0 is updating all SCSI device drivers by using the Hard Disk Setup (2.0.4 or later) utility for Apple hard drives that accompanies the System 7.0 software (for third-party devices, refer to the vendor to determine if updated drivers are necessary) BEFORE upgrading to System 7.0. The reason for this is that these newer drivers are incompatible with Virtual Memory. Failure to update these structures will prohibit using VM--it won't even turn on--however, you'll be able to use all other features of System 7.0 without any problems provided, of course, that your configuration meets the recommended memory requirements and all your software is fully compatible with this system software release.

Eliminating Macintosh VBL animated cursor ghosts

Date Written: 12/14/90

Last reviewed: 7/26/91

What do I have to do to prevent an animated cursor from leaving ghosts on the Macintosh screen?

———

Make sure the low-memory global CrsrBusy is FALSE before you try to change the cursor. MacApp does this.

Macintosh interrupt routines and I/O

Date Written: 10/1/91

Last reviewed: 10/1/91

Are there any dangers in doing a synchronous read at interrupt time? In particular, if an asynchronous write is executing and is interrupted by a synchronous read, can the re-entrancy into the Macintosh ROM read/write code cause problems?

———

The Macintosh ROMs are not re-entrant. For this and for timing considerations, you should do as little as possible within an interrupt routine. In any case, you should not be doing I/O of any kind from within your interrupt routines. You should simply use the interrupt to do the minimum possible to respond to the signaled condition and set indicators to your larger application to handle the larger context, like initiating additional I/O.

How the Macintosh mouse/cursor mechanism works

Date Written: 5/3/89

Last reviewed: 6/14/93

I'm writing a tablet (mouse, pointer, etc.) driver, and I need to be able to position the cursor and post mouseUp mouseDown events. How do I do it?

———

Here is an explanation of the Macintosh mouse/cursor mechanism, which should provide the information necessary for you to place the cursor:

The mouse is a "relative" device, which means it does not return actual coordinates; it returns a "count" or amount of movement since the last report.

When the mouse has new information, it interrupts the Macintosh. The interrupt handler adds the horizontal and vertical counts to MTemp (a low-memory location), and sets crsrNew to tell the system that the coordinates are new. Some time later (but before normal VBLs are executed) the cursor VBL task is executed, and it compares MTemp with RawMouse (which has the last value), and figures out the delta (that is, the horizontal and vertical distance moved), and does the scaling trick depending on what the user has selected in the control panel. It also updates MTemp to reflect the new value. Then it draws the cursor.

Here are the names and locations of the relevant low-memory locations:

```
RawMouse        equ $82C            ;point
MTemp           equ $828            ;point
CrsrNew         equ $8CE            ;byte
CrsrCouple      equ $8CF            ;byte
```

(NOTE: These may change in future CPUs.)

If you wish to place the cursor in an absolute location on the screen, you must set RawMouse, and MTemp to the same value, and set CrsrNew to the same value as CrsrCouple.

Keep in mind that moving the cursor from an application may violate the Human Interface Guidelines, possibly confusing and frustrating the user. Also keep in mind that the low-memory locations named above are not documented, and as such, are considered unsupported and volatile.

Posting mouse down/up events is a simple matter of keeping track of the last state of your device button, and when you detect a change, you post the appropriate event. If you wish to support the ToolBox Event Manager call Button, you must also update the low-memory global MBState. The high bit of MBState reflects the current state of the mouse button: 1 = button up, 0 = button down.

Unfortunately, the system mouse driver will change this value if it notices that the state of MBState is different from the state of the real mouse button. This means if there is a mouse connected to the Macintosh while your device is connected, it will override whatever you put in MBState. On systems equipped with ADB, this will happen only when the mouse moves, or the button is pressed. On earlier systems this happens all the time, since the button checking routine was a VBL task, and is always executed. One possible workaround is to patch Button, and return the state of your button, rather than MBState. The obvious problem is that you disable the real mouse from working with Button.

Information on the IOP Manager not available

Date Written: 6/29/90

Last reviewed: 12/17/90

After reading the preliminary release notes for the Macintosh IIfx, I found myself craving more information on the IOP interface and the IOP Manager. Where can I get more information about this new manager?

———

Apple will not be releasing IOP Manager information to developers because we plan on expanding the use of the IOPs in the future.

As the release notes stated, the IOP Manager can handle up to eight IOPs, but only two are currently used. The other six cannot be used by developers, because they are reserved by Apple for this future expansion.

How to set the Macintosh cursor at interrupt time

Date Written: 5/14/90

Last reviewed: 12/17/90

How do I set the Macintosh cursor at interrupt time, such as a VBL task?

———

Changing the cursor at interrupt time is permissible as long as the cursor handle is locked in memory and the cursor routines are not busy. A test needs to be performed before changing the cursor. If you want to call SetCursor (with a cursor handle locked in memory), you must check CrsrBusy, a low-memory global defined in MPW SysEqu. If CrsrBusy is true, then you cannot call SetCursor. Changing the cursor while CrsrBusy is true causes it to leave mouse bits, or trails, on the screen.

```
    CrsrBusy   EQU $8CD ; Cursor locked out? [byte]
```

Back to top

Downloadables

 Acrobat version of this Note (K) Download

Back to top
