New Technical Notes

Macintosh

Developer Support

Event Manager Q&As

Toolbox

Revised by: Developer Support Center June 1993
Written by: Developer Support Center October 1990

This Technical Note contains a collection of Q&As relating to a specific topic—questions you've sent the Developer Support Center (DSC) along with answers from the DSC engineers. While DSC engineers have checked the Q&A content for accuracy, the Q&A Technical Notes don't have the editing and organization of other Technical Notes. The Q&A function is to get new technical information and updates to you quickly, saving the polish for when the information migrates into reference manuals.

Q&As are now included with Technical Notes to make access to technical updates easier for you. If you have comments or suggestions about Q&A content or distribution, please let us know by sending an AppleLink to DEVFEEDBACK. Apple Partners may send technical questions about Q&A content to DEVSUPPORT for resolution.

New Q&As in this Technical Note:
Getting OSEvents from a jGNEFilter

Getting OSEvents from a jGNEFilter

Date Written: 1/19/93
Last reviewed: 4/1/93

I've written a jGNEFilter to do something whenever an application comes to the foreground. Unfortunately it seems that OSEvents aren't sent through this filter, so I never see the resumeEvent. Is this correct or am I doing something wrong? Is there a better way to get something done on a resumeEvent?

—

You're correct in that a jGNEFilter won't see OSEvents; because it sits below the Process Manager, it will see only those events that are generated by the Toolbox itself, rather than the Process Manager. The best way to ensure getting these events is probably to tail-patch GetNextEvent and WaitNextEvent; depending on how you want to deal with these events, patching EventAvail may also be necessary. There isn't a hook in between the Process Manager and the actual results of these traps, so there's no more general place to sit to do this.

Consistent application scrolling speed for all Macintosh systems

Date Written: 5/3/89
Last reviewed: 8/1/92

How can I keep the speed of my application's scrolling consistent on all Macintosh systems, including faster ones such as the Macintosh II?

—

One method is to use TickCount. If a certain number of ticks has passed since the last increment of the scroll bar, then go on. If not, then loop until the desired number of ticks has passed before scrolling. It's easy to include this test, but you'll have to play with itawhile to find the right number of ticks for the desired delay. Also, consider that some Macintosh systems are faster than a Mac II, such as the SE/30 and Macintosh systems with third-party accelerator boards. It might be nice to let users set their preference for the scrolling speed.

X-Ref:
"The Toolbox Event Manager," *Inside Macintosh* Volume I

'SIZE' resource is32BitCompatible flag
Date Written: 3/14/91
Last reviewed: 6/7/91

Does setting the is32BitCompatible bit in the 'SIZE' resource have any effect in System 7.0?

—

The alert box that was to be shown for applications with the 'SIZE' resource's is32BitCompatible flag disabled was found to be too confusing for an end user, so the is32BitCompatible flag is not used and the alert box is not displayed in the final System 7.0. (It is, however, displayed in A/UX 2.0 and 2.0.1.) This could change in the future.

'SIZE' resource bit and resume events in window title bar
Date Written: 9/3/91
Last reviewed: 9/16/91

We set our 'SIZE' resource to say we don't want mouse events for the click that generates a resume event, but when the user clicks in the title bar of our (inactive) window we get the event anyway.

—

You are quite right, you do get the mousedown if it occurs in the windows title bar. This choice was made so that when you click in the drag region of a window in the background you don't have to click twice to get the window to drag. This somewhat negates the effect that some developers are trying to get by setting the bit in the 'SIZE' resource, but it really could not be avoided. Your best bet is to simply ignore the first mousedown after a resume if it is in the drag region of the window. This way, you will not loose any events in the unlikely event that the user can get a keystroke or something in the queue before you get your

resume.

Code snippet that uses StillDown and WaitMouseUp for dragging
Date Written: 6/11/91
Last reviewed: 8/13/91

The code below gives a simple example of how to use StillDown and WaitMouseUp to do simple dragging.

```
Point oldPoint,newPt;   /* contains the old and new mouse positions */
EventRecord theEvent;   /* contains the mouseDown event */
Rect r;                 /* this has already been set by now */

   ... called with mouseDown event in theEvent ...

   oldPoint = theEvent.where;
   GlobalToLocal(&oldPoint);
   if (StillDown()) {
       while (WaitMouseUp()) {
           GetMouse(&newPoint);
           if (DeltaPoint(oldPoint,newPoint)) {
               EraseRect(&r);
               OffsetRect(&r,newPoint.h-oldPoint.h,newPoint.v-oltPt.v);
               FrameRect(&r);
               oldPoint = newPoint;
           }
       }
   }

   ...
```

Using GetKeys to check Macintosh key status

Date Written: 7/26/90
Last reviewed: 10/1/91

Is there a way to test whether a particular key is down independently of the event record? My application needs to check the Option key status before entering the main event loop.

—

The call GetKeys(VAR theKeys:KeyMap) returns a keyMap of the current state of all the keys on the keyboard. The call is documented in *Inside Macintosh* Volume I on page 259. The Option key will appear as the 58th bit (counting from 0) in the map. In MacsBug you can see this with a DM KeyMap, which returns the following:

```
0000 0000 0000 0004 0000 0000 0000 0000
```

It's important to understand that the keyMap is an array of packed bits. You need to test whether the Option key bit is 1 or 0. The key code 58 = $3A is the 58th bit of the keyMap. This number can be determined from the keyboard figure on page 251 of *Inside Macintosh* Volume I and pages 191-192 of Volume V. (If in counting the above bits you get 61 instead of 58, remember that the bits within each byte are counted right to left.)

With the above information you should be able to determine the status of any key on the keyboard within your program without waiting for an event. GetKeys, however, should be called only for special situations. Normal keyboard processing should be done through events; otherwise, your application risks incompatibilities with nonstandard input devices.

WaitNextEvent mouseRgn parameter & mouseMoved events

Date Written: 7/23/91
Last reviewed: 8/30/91

Passing nil as the mouseRgn in WaitNextEvent is entirely acceptable if a Macintosh application doesn't need to respond to changes in the location of the mouse pointer. MouseMoved events are generated only as a convenience to the programmer.

From *Inside Macintosh* Volume VI, if nil is passed as the mouseRgn, mouseMoved events will _not_ be generated. Otherwise, a mouseMoved event will be generated when the user moves the mouse outside the specified region.

Inside Macintosh Volume VI, page 5-10, has a small snippet containing a main event loop which you may find useful. It describes Apple's recommended way of calling WaitNextEvent with a non-nil mouseRgn parameter. Volume VI, page 5-29, has a long description of WaitNextEvent.

System 7 applications need to be background-capable
Date Written: 8/1/91
Last reviewed: 8/30/91

Do all System 7-savvy programs need to run with background processing enabled?

—

Yes, System 7-savvy applications should have the SIZE resource's background processing bit set. (It's not documented explicitly that you need to have this bit set.)

All System 7 Apple event-aware applications need to be background-capable, since there are many instances where Apple events will come in to you while you're in the background, and there will be many times (as new applications are developed) when you will not come to the front to process a series of events; you'll work in the background as a client for another application.

You don't want to hog a lot of system time when you have nothing to do in the background, but with the Edition Manager you do need to be able to receive events while you're in the background. However, you can still be system-friendly when you do this. Here's one way: When you're switched into the background, set your sleep time to MAXLONGINT and make sure you have an empty mouse region. This way, you'll be getting null events very rarely, and you won't be taking much time away from other applications, but you can still react to events sent to you by other parts of the system. Then when you come forward, you can reset your sleep time to your normal, low, frontmost sleep.

Note that WaitNextEvent is implemented when running System 6 without MultiFinder, but there's no DA Handler ensuring that DAs receive time. In this case, large sleep values prevent DAs from receiving timely accRun calls—the Alarm Clock DA stops ticking, for example. A compromise that doesn't hog too much processing time is to use sleep values only as large as 30-60 ticks for System 6.

Help balloons & OSEventAvail between BeginUpdate & EndUpdate
Date Written: 8/2/91
Last reviewed: 9/16/91

Random garbage is occasionally displayed on the screen if a Macintosh help balloon goes away due to an OSEventAvail call when a screen refresh is taking place. Is it possible that the Help Manager code isn't entirely cool if it hides a balloon between a BeginUpdate and EndUpdate?

—

Yes, this is particularly nasty. Here's the deal:

OSEventAvail does do balloon maintenance. This is a problem, because OSEventAvail is documented as to NOT move memory, but when balloons are active, it does sometimes. Beyond this, which is really a bad thing, it is still a problem for updates that allow interrupting, such as yours.

Your problem isn't that it moves memory. What it does do is recalculate visRgns, due to the balloon moving. The balloons are actually windows, and the Help Manager has its own WDEF that does the funny balloon shapes. When a balloon/window is moved, the visRgns have to be recalculated to reflect what has been covered or exposed. This is normally fine, but it is a really bad thing between BeginUpdate and EndUpdate.

BeginUpdate first caches the visRgn. It then localizes the updateRgn. It then intersects the visRgn and updateRgn and places the result in the visRgn. It then clears the updateRgn. You are now ready to update just the portion of thewindow that needs updating.

The problem is that you are making a call to OSEventAvail between BeginUpdate and EndUpdate with balloons active, and somebody moves the mouse. The balloon therefore moves, the visRgns are recalculated, and the window being updated now has a new visRgn.

This would not be bad, except that the visRgn that was cached when BeginUpdatewas called reflects the balloon's old position. When EndUpdate is called, the visRgn for the window just updated will be restored to what it was before the balloon moved.

This now means that there is possibly an area on the screen that is shared between the updated window and the balloon window, plus a possible area that should be in a window, and is no longer. Obviously, this is really bad.

The only thing that I can think of to do is the following:

1) Call BeginUpdate, as usual.
2) Copy the resultant visRgn into the clipRgn.
3) Call EndUpdate
4) Draw to the window as you would for a normal update. (Your normal update also calls OSEventAvail every so often to see if it should interrupt.)

What this accomplishes is that the clipping for the area that doesn't need updating is no longer done with the visRgn. It is done with the clipRgn instead. This will not be affected by

any balloons moving. Also, after the EndUpdate, there is no longer a cached copy of the window's visRgn, and therefore the balloon window can move and not mess up visRgns.

The above general technique should take care of the problem for you. At this point, there isn't a resolution to this problem -- only a workaround. It is bad that OSEventAvail can move memory, as it was documented not to. The problem is that the Help Manager needs to do stuff at this time, and this may involve moving the balloon window.

DTS recommends making your code safe from this problem with the above workaround, and to also see if you can be hurt by memory moving when OSEventAvail is called in general.

Macintosh Finder and DoubleTime global
Date Written: 8/23/91
Last reviewed: 10/8/91

How does the Macintosh Finder interpret the DoubleTime global? It seems the Finder doesn't use DoubleTime the way it's documented in *Inside Macintosh.*

—

The ratio of ticks to value of DoubleTime is 1:1—that is, the number in the DoubleTime variable *(Inside Macintosh* Volume I, page 260) is, in fact, the number of ticks between a mouse up and a mouse down. Of course, this is not how the Finder works. The Finder multiplies the DoubleTime variable by 2 to determine double click time. It does this to account for user problems that occur while the user is arranging icons. Thus, the hard-and-fast answer is the Finder uses DoubleTime*2, and the rest of the system just uses DoubleTime.

By the way, the Finder does not limit the double-time variable to 64 ticks. It treats it like a byte in most places, although in some places it is treated like a longword. However, clipping it at 64 ticks would be the best method since that would provide a 1 second double click (two second in the Finder), which is long enough for anyone.

Applications should use DoubleTime as documented in the manual, not as it's used by the Finder.

Events and switching between Macintosh applications
Date Written: 830/91
Last reviewed: 10/22/91

After an application in the System 7 Application or Apple menu is selected, sometimes control doesn't switch from our application to the selected application. What could be wrong and how can I zero in on the problem?

—

The symptoms you've described—the process menu not switching layers—is exactly what happens if you have an activate or update event pending that you're not acting on. Here are ways that pending activate or update events can be handled incorrectly:

• You're not calling BeginUpdate and EndUpdate (the most likely problem).

• Your application isn't asking for all events.

• Your application has dueling event loops.

• The word in your code that contains the everyEvent constant is trashed (unlikely).

PostHighLevelEvent and sending low-level messages
Date Written: 8/23/91

Last reviewed: 9/15/92

It looks as though the Event Manager routine PostHighLevelEvent could be (ab)used to send low-level messages, like phony mouse clicks and keystrokes. Would this work?

—

No; unfortunately, this won't work. A few reasons why:

• The only applications that will receive high-level events (and their descendants, like Apple events) are applications that have their HLE bit set in their SIZE resource. If you try to send (or post) an HLE to an older application you'll get an error from the PPC Toolbox telling you that there's no port available.

• There's no system-level translator to convert these things. There are currently translators to change some Apple events. Specifically, the Finder will translate any "puppet string" event into puppet strings for non-System 7 applications (odoc, pdoc, and quit), but these are very special.

• The only way to send user-level events such as mouse clicks through HLEs is to use the Apple events in the MiscStndSuite shown in the Apple Event Registry. And all those events assume that the receiving application will do the actual translations to user actions themselves.

• HLEs come in through the event loop. So even if it were possible (through some very nasty patching to WaitNextEvent) to force an HLE into a non–HLE-aware application, the event would come in with an event code of 23 (kHighLevel) and the targeted application would just throw it away.

So the answer is that you can't send user-level events to an HLE-aware application. If you want to drive the interface of an old application in System 7, you have to use the same hacky method you used under all previous systems. This, by the way, is one of the main reasons why MacroMaker wasn't revised for System 7. Apple decided that it wasn't supportable and that we would wait for applications to update to System 7 and take advantage of third-party Apple event scripting systems.