New Technical Notes

Macintosh ®

Developer Support

ME 505 - Memory Manager Q&As

Memory

Revised by: Developer Support Center September 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:
Lock GetPhysical address range even with VM disabled
Caching details for locked pages on 68030 and 68040

Lock GetPhysical address range even with VM disabled

Date Written: 3/10/93
Last reviewed: 6/24/93

Page 318 of Inside Macintosh: Memory says you must lock the address range passed to the GetPhysical routine to guarantee that the translation data are accurate (that is, paging activity hasn't invalidated translation data). Is this true even if Virtual Memory is disabled? Does the Macintosh Operating System ever change the mapping of physical to logical addresses when VM is really off?

—

In the current implementation of the OS, pages don't move when VM isn't on; however, we can't guarantee that that will always be true. Also, the GetPhysical implementation may not always allow you to translate addresses that haven't been locked (currently, it appears to allow this, but this may also change). So, you should lock the GetPhysical address range even with VM disabled.

Caching details for locked pages on 68030 and 68040

Date Written: 3/10/93

Last reviewed: 6/24/93

The Macintosh Technical Note “Cache as Cache Can” states that data cacheing may be disabled or pages may be marked noncacheable in certain cases. Could you provide details of what exactly happens in 68030 and ‘040 CPU types. Which option is taken and what criteria is used in each case?

—

Basically, the caching situation for locked pages is this: on the Macintosh IIci and IIsi, locking a page disables the cache as a fix for a hardware problem. On other 68030 machines, the cache will be disabled with a 1 megabyte page resolution; on 040 machines, the page size is 16K. (***Tim, add “when VM is off”?***)

Validating handles passed from another application

Date Written: 1/14/93
Last reviewed: 6/14/93

I’ve written a control that is passed a handle from an application for which I don’t have the source. This application sometimes passes invalid handles. I check to make sure a handle and its pointer are even and call GetHandleSize to verify the handle. This works about 95 percent of the time, but sometimes an invalid handle causes GetHandleSize to crash. How can I verify a handle?

—

As you’ve found, passing a bogus handle to most Memory Manager calls will cause a deadly crash, which helps make validating handles a challenge. Even when you can determine that a long value is a valid handle, there’s always the possibility that it doesn’t point to data that you want or should be messing with. For example, you could be setting the size of a handle that belongs to a resource, completely confusing the Resource Manager. A valid handle pointing to someone else’s data may make you crash after you try to use the data. You must also deal with the possibility that the handle is correct but your code doesn’t like it because it’s been allocated in the application’s secondary heap, or it may be in the system heap or temporary memory. There’s also a performance hit when you take time to validate handles.

In short, it’s better to change the source of the invalid handles so that it stops passing you garbage than to deal with validating handles. If you must do it, here are some checks that will help you get some certainty on the validity of a handle you receive:

• A handle is an multiple of four.
• A handle is between the top of the system heap and bufPtr (within useful RAM).
• If the heap is known, walk the heap or just check that the address is in bounds.
• Repeat checks on handle^ to make sure.

Again, there’s no way to determine whether a random value is the handle you expect to receive. The best documentation on the subject is the *Inside Macintosh: Memory;* it describes the structure of zones and heaps in depth. Check it out for more precise details on how to implement the validation code if you decide to go ahead with it.

GetMHandle warning applies only to popUpCDEF menus

Date Written: 1/14/93

Last reviewed: 3/10/93

The System 7.1 Digest has a disturbing comment about GetMHandle—namely that it was never supported and will no longer work. Is this true?

—

This warning is misleading and is being corrected in future release notes. It applies *only* to pop-up menus created with the pop-up menu control. Before System 7.1, after a control was created, GetMHandle would return the menu handle for the control although it was never documented as doing so. In System 7.1 it was changed so that the menu would be inserted into the menu list only when the control was getting ready to pop up the menu and deleted as soon as the control was done with it, so you could no longer use GetMHandle to retrieve the menu handle. The proper way to get the menu handle is from the mHandle field of the popupPrivateData structure. The handle to this structure is in the contrlData field of the pop-up menu's control record.

A corollary is that the pop-up control has always checked to see if the menu was already in the menu list. If it is, the control doesn't get the menu from the menu resource and doesn't delete the menu when it's done. You can use this feature, for example, if you want to create a menu with NewMenu rather than getting it from a resource. In this case, and all other cases where the application inserts and deletes the pop-up menu in the menu list itself, GetMHandle can be used to retrieve the menu handle because it 's under the control of the application.

Checking for GetHandleSize error conditions

Date Written: 10/14/91
Last reviewed: 6/14/93

Inside Macintosh Volume II, page 33, states that _GetHandleSize returns D0.L >= 0 if the trap is successful or D0.W < 0 if the trap is unsuccessful. What happens if the handle size is 0xFFFF, for instance? A TST.W will indicate an error when in fact there is none. How should I check for this condition?

—

Inside Macintosh is correct (although confusing) regarding the determination of an error condition. The way to do it is first test the long to see if it's valid (D0 >= 0). If the long is valid, you can continue with confidence that no error occurred. If, however, the long in D0 is negative, the low word contains the error (and currently the high word contains $FFFF, the sign extension). The reason the manual highlights the fact that only the low word contains the error is to allow you to save the error in standard fashion since all other errors are word sized, and also to caution you against using the processor status on exit from GetHandleSize since it will be based on the low word only. In other words, if the long is negative, simply ignore the high word. Here's some assembly code that will work:

```
      move.L    theHandle(a6),A0
      _GetHandleSize
      tst.L     D0
      bpl.s     @valueOK
      move.W    D0,theError(A5)
      moveQ     #0,D0
@valueOK
```

How Macintosh memory location 0 gets changed

Date Written: 3/12/91
Last reviewed: 6/14/93

Why does the longword at location $0 get changed to 0x40810000 at every trap?

—

In System 7, the Process Manager slams a benign value into location $0 to help protect against bus errors when an application inadvertently dereferences a NIL pointer. (There's no bus-error checking on writes to ROM, so the "benign value" is usually ROMBase+$10000.)

If you're debugging, you want the opposite effect: you want these inadvertent accesses to "cause" bus errors. If you put a different value in location $0 before the Process Manager starts up (that is, from MacsBug or TMON initialization, or from an INIT like EvenBetterBusError), it will force that value instead. For more information, see the "Macintosh Debugging" article in this issue.