New Technical Notes

Macintosh

Developer Support

Memory Manager Q&As

Memory M.ME.MemMgr.Q&As

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:
Validating handles passed from another application

Validating handles passed from another application
Date Written: 1/14/93
Last reviewed: 4/1/93

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: 11/6/91

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: 10/9/91

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.

VBL tasks shouldn't call Macintosh Memory Manager
Date Written: 10/29/90
Last reviewed: 8/1/92

What's wrong with having VBL (Vertical Blanking) tasks make calls to the Macintosh Memory Manager, either directly or indirectly? The problem is that the Memory Manager could be moving memory around when an interrupt occurs. If the VBL task also moves memory, the heap could be destroyed.

—

VBL tasks are intended to provide a method of time-syncing to the video beam of the display. (On slotted Macintosh models you'd use SlotVInstall.) They're also used to get periodic time for short tasks, although the Time Manager is better for this. VBL tasks should minimize execution time. The best use of a VBL task is to do a short condition check and set a flag for the main process to indicate that it's now a good time to do something.

Memory Manager ROM bug and MMInit
Date Written: 2/28/91
Last reviewed: 6/30/91

Where can we get our hands on a fix to the ROM bug in the Macintosh IIci and Macintosh IIfx Memory Manager? Word has it that Apple wrote an INIT (MMInit) to fix this problem. Because we're using our application on both the Macintosh IIci and the IIfx, and the application uses many handles, this fix would be appreciated.

—

Under System 6.0.x, Apple had identified a minor problem with the Memory Manager in the Macintosh IIfx, IIci, IIsi, and LC. This problem resulted in a performance degradation in an extremely small number of applications and did not cause system crashes. We believe this was an insignificant problem that affected few applications and very few customers. The problem was not new and was not caused by the introduction of System Software 6.0.5 or 6.0.7.

Based on developer feedback, customer feedback, and extensive in-house testing, Apple has identified very few affected applications. Because the problem affects only an extremely small number of developers, Apple is working with those developers to fix their applications. The best solution, however, is to upgrade to System 7.0, as it includes an enhancement to the Memory Manager to address this issue.

During extensive testing and research, Apple investigated a variety of solutions to enhance the Memory Manager performance. One area researched was a software solution called the Memory Manager INIT (a software "patch"), or MMInit as it's most often called. Through testing we discovered that this patch did not enhance Memory Manager performance and introduced risks such as decreased performance in some mainstream applications.

An unofficial version of the Memory Manager INIT has surfaced. This INIT should not be used, because it has been modified from Apple's experimental version and could cause data corruption, data loss, and crashes. Apple strongly urges that you discard the INIT if you have obtained a copy, and not use it. If any version of this INIT is used under System 7.0, it defeats the enhanced Memory Manager and reintroduces the bugs that were present in the System 6.0.x Memory Manager.

You may still wonder if you have been affected by this problem and how to avoid it under 6.0.x. The problem is most severe when allocating pointers in a heap with a rather large number of handles (on the order of tens of thousands). It's helpful to allocate enough master pointers (via MoreMasters) during initialization. If the Memory Manager has to call MoreMasters later on, not only could it fragment memory, but it could take an exceedingly long time. It's also helpful not to allocate thousands of handles. Besides requiring lots of master pointers, it takes the Memory Manager a long time to crunch through them during

heap compaction.