Technical Note IMERRATA02
Inside Macintosh: Memory Errata

CONTENTS

Topics

About This Book

Chapter 1 - Introduction to Memory Management

Chapter 2 - Memory Manager

Chapter 3 - Virtual Memory Manager

Chapter 4 - Memory Management Utilities

References

Downloadables

This Technote discusses known errors and omissions in *Inside Macintosh: Memory.*

[Aug 01 1998]

Topics

- *Inside Macintosh:Memory* doesn't apply to 64K ROM Macintoshes, August 1998
- `TickCount` vs. `Ticks`, August 1998
- Correction to Description of Figure 1-7, October 1994
- Correction to Documented Default Stack Sizes, October 1994
- Addendum to Description of `SetApplLimit`, October 1994
- Correction to Return Value Type, October 1994
- Correction to `NewPtr` Description, October 1994
- Correction to Temporary Memory Locking Requirements, August 1998
- Correction to Listing for Checking for Temporary Memory Routines, October 1994
- Clarification of `'sysz'` Resources and System Extensions, August 1998
- Clarification of Block Alignment Boundaries, October 1994
- Clarification of Block Header Diagrams, October 1994
- Clarification of `PtrToHand` description, August 1998
- Calling conventions for `PtrToHand`, `PtrToXHand`, and `PtrAndHand`, August 1998
- Correction to `HandToHand` description, August 1998
- `HandAndHand` warning no longer needed, August 1998
- Correction to `PtrAndHand` Assembly-Language Information, August 1998
- Addendum to `StackSpace` Special Considerations, August 1998
- Addendum to `MemError` Warning, August 1998
- Addition to `MyGrowZone` and `MyPurgeProc` Special Considerations, August 1998
- Correction to `LockMemoryContiguous` Description, October 1994
- Correction to `DebuggerGetMax` Description, October 1994
- Correction to `DebuggerLockMemory` Description, October 1994
- Correction to `DebuggerUnlockMemory` Description, October 1994
- Finding QuickDraw globals, August 1998

Back to top

About This Book

***Inside Macintosh:Memory* doesn't apply to 64K ROM Macintoshes**

Page xi

This note should be added to the introduction:

The intent of *Inside Macintosh:Memory* is to cover all Macintosh models existing or foreseeable at time of print, except for a few early models with the 64K ROMs, which are no longer supported:

- The original Macintosh (64KB ROM, 128KB, or 512KB RAM)
- Macintosh XL (Lisa hardware emulating an original Macintosh)

Inside Macintosh:Memory does apply to the Macintosh 512K enhanced (featuring the same 128K ROM as the Macintosh Plus) and all Macintoshes at time of print, including Performa, PowerBook, Power Macintosh, and Mac OS-compatible PowerPC machines.

Back to top

Chapter 1 - Introduction to Memory Management

`TickCount` vs. `Ticks`

Pages 1-6 and 2-7

This is a bad example. A better example is `MemError` vs. `MemErr`.

The paragraph on page 1-6 should read:

In general, it is best to avoid reading or writing low-memory system global variables. Most of these variables are undocumented, and the results of changing their values can be unpredictable. Usually, when the value of a low-memory global variable is likely to be useful to applications, the system software provides a routine that you can use to read or write that value. For example, you can get the current value of the `MemErr` global variable by calling the `MemError` function.

The paragraph on page 2-7 should read:

Even when *Inside Macintosh* does document a particular system global variable, you should use any available routines to access that variable's value instead of examining it directly. For example, you should use the `MemError` function to find the last error returned by the Memory Manager instead of examining the `MemErr` global variable directly.

Correction to Description of Figure 1-7

Page 1-14

The description of Figure 1-7 is incorrect. The paragraph immediately following Figure 1-7 should read: "In Figure 1-7, Application 2 has almost exhausted its application heap. As a result, it has requested and received a large block of temporary memory. Application 2 can use the temporary memory in whatever manner it desires."

Correction to Documented Default Stack Sizes

Pages 1-39 through 1-40, Changing the Size of the Stack

The documentation states that by default the stack can grow to "32KB on computers with Color QuickDraw" (page 1-40). The actual default stack size on machines with Color QuickDraw is 24KB. In addition, the default stack size for a background-only application (or a "faceless background process") is 2KB, not 8KB as claimed on page 1-40.

However, the default stack size may change in the future and should never be assumed.

Addendum to Description of `SetApplLimit`

Pages 1-39, 1-52, and 2-84.

The description of the `SetApplLimit` routine should mention that `SetApplLimit` enforces a minimum stack size equal to `DefltStack`. For example, calling `SetApplLimit` on a background-only application running on a Color QuickDraw machine will result in a minimum stack size of 24KB, regardless of the value passed to `SetApplLimit`. The partition of the background-only application must be large enough to accommodate a stack of that size.

Correction to Return Value Type

Page 1-45, Allocating Blocks of Memory

The application-defined function `NewPtrCushion` (defined in Listing 1-7) should return a value of type `Ptr`, not of type `Handle`.

Correction to `NewPtr` Description

Pages 1-58 through 1-59, and 2-36.

The Assembly-Language Information for the `NewPtr` function states that on entry register `A0` contains the number of logical bytes requested. In fact, register `D0` should contain the number of logical bytes requested.

Addition to `SetCurrentA5` Description

Pages 1-79 and 1-81

There is a problem introduced with MultiFinder. It is well documented in Technote ME14-The New Memory Manager and You, under A5 World Problems and Heap Callback Procedures (but the workaround suggested in this technote is unusable).

The problem: when a heap is low on memory or a `Handle` is going the be purged, the `GrowZone` or `PurgeProc` function is called without an `A5` world switch. This means that the `GrowZone` or `PurgeProc` function is called with `CurrentA5` set, according to the `A5` world of the application that did the call generating the low-memory or `Handle` purge condition, rather than the value of `A5` for which the application the `GrowZone` or `PurgeProc` function belongs. As the Technote says, this "[will] cause all hell to break loose". This may occur, for example, when the active and frontmost application closes a window. The update region of windows belonging to other applications need to be updated, which often involves extending the update region `Handle`, which can cause a `GrowZone` procedure to be called, but since the window being updated is in the background, the value of `A5` is wrong with respect to the `GrowZone` or `PurgeProc` function.

The safe workaround, which is needed only in 680x0 code, involves saving `A5` explicitly, and recovering it using PC-relative addressing. Going forward to Mac OS X and Carbon, `GrowZone` and `PurgeProc` functions will not be needed, so this problem will gradually go away.

Back to top

Chapter 2 - Memory Manager

Correction to Temporary Memory Locking Requirement

Page 2-10, Extending an Application's Memory

The phrase:

> [...] you must never lock temporary memory across calls to `GetNextEvent` or `WaitNextEvent` [...]

should read:

> [...] you should avoid locking temporary memory across calls to `GetNextEvent` or `WaitNextEvent` [...]

See Q&A ME08-Temporary Memory for details.

Correction to Listing for Checking for Temporary Memory Routines

Page 2-12

Listing 2-3 on page 2-12 contains an error. The line:

```
TempMemCallsAvailable := BAND(myRsp, gestaltTempMemSupport) <> 0;
```

should be replaced by the line:

```
TempMemCallsAvailable := BTst(myRsp, gestaltTempMemSupport);
```

Clarification of `'sysz'` Resources and System Extensions

Page 2-13, Allocating Memory at Startup Time

The documentation implies (page 2-13) that system extensions running under System 7.0 and later do not need to have `'sysz'` resources to indicate their system heap memory requirements. This statement is misleading. `'sysz'` resources are effective on all versions of system software.

Furthermore, the generic term "system extension" is ambiguous in this context. Many items in the Extensions folder, such as components and native drivers (`'ndrv'`s), are labeled as "system extensions" by the Finder. However, `'sysz'` resources are only applicable to files that may contain `'INIT'` resources. This includes control panels, system extensions of file type `'INIT'`, and application extensions with an `'INIT'` resource. [Radar ID 1245268]

Before executing an `'INIT'` resource from a file, the Start Manager first checks to see whether the system heap has *X* bytes of free space, where *X* is the maximum of 16KB or the value in the file's `'sysz'` resource. If less than *X* bytes of free space is available, the Start Manager grows the system heap such that *X* bytes are available for the `'INIT'`s use.

Note that in System 7.0 and later, the system heap can grow after the startup process has completed. The `'sysz'` resource only controls the system heap expansion at startup time.

Clarification of Block Alignment Boundaries

Page 2-22, Block Headers

The documentation states that "on computers containing the MC68020, MC68030, or MC68040 microprocessors, blocks are padded to 4-byte boundaries." This is incorrect for 68040 and PowerPC machines, where blocks are always aligned on 16-byte boundaries.

Starting with Mac OS 7.5, all memory blocks, regardless of the runtime CPU, are aligned to 16-byte boundaries.

Clarification of Block Header Diagrams

Pages 2-22 through 2-23, Block Headers

Figures 2-1 (page 2-22) and 2-2 (page 2-23) might be misleading. Remember that *Inside Macintosh* typically draws memory diagrams with the low-memory addresses nearest to the bottom of the diagram. (Compare Figure 1-1 on page 1-5, where the low-memory end of the diagram is explicitly labeled.) The long word containing the block type and size correction is the first long word in the 24-bit zone header. Similarly, the long word containing the block type and unused space is the first long word in the 32-bit zone header.

Figure 2-2 (page 2-23) claims that a block type of "11" signifies a relocatable block. This is incorrect. The correct type indicator for a relocatable block is "10".

Clarification to `PtrToHand` Description

Page 2-61

The `dstHndl` parameter is not an input parameter; therefore, the statement "The `dstHndl` parameter must be a handle variable that is not empty and is not a handle to an allocated block of size 0" is misleading in that it implies that you may have to allocate the handle yourself. This is not the case. `PtrToHand` will allocate a handle of the requested size and return it to you in the `dstHndl` parameter. If no error occurs, on exit `dstHndl` is an unlocked, non-purgeable `Handle` of the requested size, allocated from the current heap zone.

Correction to `HandToHand` description

Page 2-63

The statement:

> "The new relocatable block is created in the same heap zone as the original block (which might not be the current heap zone)."

is incorrect.

The correct description is that the new relocatable block is created in the current heap zone, regardless of the zone of the original relocatable block. During `HandToHand`, the original `Handle` is made non-purgeable while the new one is allocated. The original `Handle` is then returned to its original state. It is safe to call `HandToHand` on purgeable handles, but not on a purged handle.

`HandAndHand` warning no longer needed

Page 2-64

The warning is no longer needed and only applied to Macs which use the original 64K ROMs.

It is safe to call `HandAndHand` on purgeable handles, but not on handles that have been purged.

Correction to `PtrAndHand` Assembly-Language Information

Page 2-65

The description for A1's value should read "Handle to relocatable block to which the data will be appended."

size, the "Number of bytes to append", is passed in `D0`, not `A2`.

Calling conventions for `PtrToHand`, `PtrToXHand`, and `PtrAndHand`

Page 2-66

Before Assessing Memory Conditions, there should be this note:

In the description of `PtrToHand`, `PtrToXHand`, and `PtrAndHand`, `Ptr` or "pointer" refers to a memory address, not necessarily to a Memory Manager `Ptr` as returned by `NewPtr`. `Handle` does refer to a Memory Manager `Handle`, allocated by routines such as `NewHandle` or `NewEmptyHandle`.

Addendum to `StackSpace` Special Considerations

Page 2-70

You should not call `StackSpace` at interrupt time, because it sets the value returned by `MemError` and therefore may cause the interrupted application to fail.

Addendum to `MemError` Warning

Page 2-70

In 680x0 code, calling a function in an unloaded segment will zero `MemErr` when the segment loader loads the segment into memory, causing `MemError` to return `noErr`. Therefore, you should always call `MemError` right after the Memory Manager function that may have caused an error.

Addition to `MyGrowZone` and `MyPurgeProc` Special Considerations

Page 2-90 and 2-91

There is a problem introduced with MultiFinder. It is well documented in Technote ME14-The New Memory Manager and You, under `A5` World Problems and Heap Callback Procedures (but the workaround suggested in this technote is unusable).

The problem: when a heap is low on memory or a `Handle` is going the be purged, the `GrowZone` or `PurgeProc` function is called without an `A5` world switch. This means that the `GrowZone` or `PurgeProc` function is called with `CurrentA5` set according to the `A5` world of the application that did the call generating the low-memory or `Handle` purge condition, rather than the value of `A5` for which the application the `GrowZone` or `PurgeProc` function belongs. As the Technote says, this "[will] cause all hell to break loose". This may occur, for example, when the active and frontmost application closes a window. The update region of windows belonging to other applications need to be updated, which often involves extending the update region `Handle`, which can cause a `GrowZone` procedure to be called, but since the window being updated is in the background, the value of `A5` is wrong with respect to the `GrowZone` or `PurgeProc` function.

The safe workaround, which is needed only in 680x0 code, involves saving `A5` explicitly, and recovering it using PC-relative addressing. Going forward to Mac OS X and Carbon, `GrowZone` and `PurgeProc` functions will not be needed, so this problem will gradually go away.

Back to top

Chapter 3 - Virtual Memory Manager

Correction to `LockMemoryContiguous` Description

Pages 3-29 through 3-30

The Assembly-Language Information for the `LockMemoryContiguous` function states that on entry register `A1` contains the "number of bytes to unlock." In fact, register `A1` contains the number of bytes to lock.

Correction to `DebuggerGetMax` Description

Page 3-34

The description of `DebuggerGetMax` states (page 3-34): "Of course, you should use the `Gestalt` function to check whether virtual memory is available at all before you call the `DebuggerGetMax` function." To see if virtual memory is available from a debugger, it is preferable to check to see whether the `_DebugUtil` trap is available rather than call `Gestalt`.

Correction to `DebuggerLockMemory` Description

Page 3-37

The Assembly-Language Information for the `DebuggerLockMemory` function (page 3-37) lists the trap macro as `_DebuggerLockMemory`. The correct trap macro should be `_DebugUtil`. Furthermore, the Assembly-Language Information for the `DebuggerLockMemory` function states that on entry register `A1` contains the "number of bytes to hold." In fact, register `A1` contains the number of bytes to lock.

Correction to `DebuggerUnlockMemory` Description

Page 3-38

The Assembly-Language Information for the `DebuggerUnlockMemory` function (page 3-38) states that on entry register `A1` contains the "number of bytes to hold." In fact, register `A1` contains the number of bytes to unlock.

Back to top

Chapter 4 - Memory Management Utilities

Finding QuickDraw globals

Page 4-18

The sentence:

"However, the `A5` register always points to the last of these global variables, `thePort`."

should read:

"However, the `A5` register always contains the address of a pointer to the last of these global variables, `thePort`."

Back to top

References

Inside Macintosh: Memory

Back to top

Downloadables

	Acrobat version of this Note (K)	Download

Back to top