New Technical Notes

Macintosh

Developer Support

Compatibility Guidelines

Overview **M.OV.Compatibility**

Revised by: Louella Pizzuti February 1987
Written by: Cary Clark January 1986
Scott Knaster

Apple has many enhancements planned for the Macintosh family of computers. To help ensure your software's compatibility with these enhancements, check each item in this note to be sure that you're following the recommendations.

If your software is written in a high-level language like Pascal or C and if you adhere to the guidelines listed in *Inside Macintosh*, many of the questions in this note won't concern you. If you develop in assembly language, you should read each question carefully. If you answer any question "yes," your software may encounter difficulty running on future Macintosh computers, and you should take the recommended action to change your software.

Do you depend on 68000 instructions which require that the processor be in supervisor mode?

In general, your software should not include instructions which depend on supervisor mode. These include modifying the contents of the status register. Most programs which modify the status register are only changing the Condition Code Register (CCR) half of the status register, so an instruction which addresses the CCR will work fine. Also, your software should not use the User Stack Pointer (USP) or turn interrupts on and off.

Do you have code which executes in response to an exception and relies on the position of data in the exception's local stack frame?

Exception stack frames vary on different microprocessors in the 68000 family, some of which may be used in future Macintosh computers. You should avoid using the `TRAP` instruction. **Note:** You can determine which microprocessor is installed by examining the low-memory global `CPUFlag` (a byte at `$12F`). These are the values:

CPUFlag	microprocessor
$00	68000
$01	68010

$02	68020
$03	68030

Do you use low-memory globals not documented in *Inside Macintosh*?

Other microprocessors in the 68000 family use the exception vectors in locations `$0` through `$FF` in different ways. No undocumented location below the system heap (`$100` through `$13FF`) is guaranteed to be available for use in future systems.

Do you make assumptions about the file system which are not consistent with both the original Macintosh File System and the Hierarchical File System?

Your applications should be compatible with both file systems. The easiest way to do this is to stick to the old files system trap calls (which work with both file systems) and avoid direct manipulation of data structures such as file control blocks and volume control blocks whenever possible.

Do you depend on the system or application heaps starting at a hard-coded address?

The starting addresses and the size of the system and application heaps has already changed (Macintosh vs. Macintosh Plus) and will change again in the future. Use the global `ApplZone` to find the application heap and `SysZone` to find the system heap. Also, don't count on the application heap zone starting at an address less than 65536 (that is, a system heap smaller than 64K).

Do you look through the system's queues directly?

In general, you should avoid examining queue elements directly. Instead, use the Operating System calls to manipulate queue elements.

Do you directly address memory-mapped hardware such as the VIA, the SCC, or the IWM?

You should avoid accessing this memory directly and use trap calls instead (disk driver, serial driver, etc.). Future machines may include a memory management unit (MMU) which may prevent access to memory-mapped hardware. Also, these memory-mapped devices may not be present on future machines. The addresses of these devices are likely to change, so if you must access the hardware directly, get the base address of the device from the appropriate low-memory global (obtainable from includes and interface files):

device	**global**
VIA	$1D4
SCCRd	$1D8
SCCWr	$1DC
IWM	$1E0

Do you assume the location or size of the screen?

The location, size, and bit depth of the screen is different in various machines. You can determine its location and size by examining the QuickDraw global variable `screenBits` on machines without Color QuickDraw. On machines with Color QuickDraw, the device list, described in the Graphics Devices chapter of *Inside Macintosh*, tells the location and size and bit depth of each screen, `screenBits` contains the location and size of the main device, and `GrayRgn` contains a region describing the shape and size of the desktop.

Does your software fail on some Macintosh models or on A/UX?

If so, you should determine the reason. Failure to run on all versions of the Macintosh may indicate problems which will prevent your software from working on future machines. Failture to run on A/UX, Apple's Unix for the Macintosh, also may indicate such problems.

Do you change master pointer flags of relocatable blocks directly with BSET or BCLR instructions?

In the future and on A/UX, all 32 bits of a master pointer may be used, with the flags byte moved elsewhere. Use the Memory Manager calls `HPurge`, `HNoPurge`, `HLock`, `HUnlock`, `HSetRBit`, `HClrRBit`, `HGetState`, and `HSetState` to manipulate the master pointer flags. (See the Memory Manager chapter of *Inside Macintosh Volume IV* for information on these calls.)

Do you check for 128K, 512K, and 1M RAM sizes?

You should be flexible enough to allow for non-standard memory sizes. This will allow your software to work in environments like MultiFinder.

Is your software incompatible with a third-party vendor's hardware?

If so, the incompatibility may prevent your software from working on future machines. You should research the incompatibility and try to determine a solution.

Do you rely on system resources being in RAM?

On most of our systems, some system resources are in ROM. You should not assume, for example, that you can regain RAM space by releasing system resources.

Does your software have timing-sensitive code?

Various Macintoshes run at different clock speeds, so timing loops will be invalid. You can use the trap call `Delay` for timing, or you can examine the global variable `Ticks`.

Do you have code which writes to addresses within the code itself?

A memory management unit (MMU) may one day prevent code from writing to addresses within code memory. Also, some microprocessors in the 68000 family cache code as it's encountered. Your data blocks should be allocated on the stack or in heap blocks separate from the code, and your code should not modify itself.

Do you rely on keyboard key codes rather than ASCII codes?

The various keyboards are slightly different; future keyboards may be different from them. For textual input, you should read ASCII codes rather than key codes.

Do you rely on the format of packed addresses in the trap dispatch table?

The trap dispatch table is different on various Macintoshes. There's no guarantee of the trap table's format in the future. You should use the system calls `GetTrapAddress` and `SetTrapAddress` to manipulate the trap dispatch table.

Do you use the Resource Manager calls `AddReference` or `RmveReference`?

These calls have been removed from the 128K ROM. They are no longer supported.

Do you store information in the application parameters area (the 32 bytes between the application and unit globals and the jump table)?

This space is reserved for use by Apple.

Do you depend on values in registers after a trap call, other than those documented in *Inside Macintosh*?

These values aren't guaranteed. The register conventions documented in *Inside Macintosh* will, of course, be supported. Often, you may not realize that your code is depending on these undocumented values, so check your register usage carefully.

Do you use the IMMED bit in File Manager calls?

This bit, which was documented in early versions of *Inside Macintosh* as a special form of File Manager call, actually did nothing for File Manager calls, and was used only for Device Manager calls. With the advent of the Hierarchical File System, this bit indicates that the call has a parameter block with hierarchical information.

Do you make assumptions about the number and size of disk drives?

There are now five sizes of Apple disks for the Macintosh (400K, 800K, and 20M, 40M, 80M), as well as many more from third-party vendors. You should use Standard File and File Manager calls to determine the number and size of disk drives.

Do you depend on alternate (page 2) sound or video buffers?

Some Macintoshes do not support alternate sound and video buffers.

Do you print by sending ASCII directly to the printer driver?

To retain compatibility with both locally-connected and AppleTalk-connected printers, you should print using Printing Managerr, as documented in *Inside Macintosh*.

Does your application fail when it's the startup application (i.e., without the Finder being run first)?

If so, you're probably not initializing a variable. If your application does not work as the startup application, you should determine why and fix the problem, since it may cause your application to fail in the future.