New Technical Notes

Macintosh

Developer Support

The Joy Of Being 32-Bit Clean

Overview

Revised by: Andrew Shebanow June 1989
Written by: Andrew Shebanow October 1988

What to do (and what **not** to do) to make your programs run under A/UX and future versions of the Macintosh System Software.
Changes since October 1988: Added information on writing 32-bit clean CDEFs, and updated A/UX information to reflect the capabilities of A/UX 1.1.

Introduction

Many programs available today will not run in a 32-bit world. Currently the Macintosh OS runs in a 24-bit world, where the hardware ignores the high byte of all memory addresses (including pointers and handles). Under A/UX (and future versions of the Macintosh OS), programs must run in a 32-bit world, where the entire address is significant. This Technical Note presents guidelines which you should follow to make your program work under A/UX and future versions of the Macintosh OS. Following these guidelines means a little extra work, but it is this extra work now which will bring you the joy of being 32-bit clean when the world changes and you don't have to rewrite your program.

Note: Much of the information presented here has already been discussed in one or more of the documents referenced at the end of this Note, but it is being repeated here because of the importance of the subject matter.

Keep in mind that the rules presented here are not graven in stone. Although you may find it necessary to break some of these rules to achieve specific functionality in your program, it is important to remember that in doing so, you will cause your program to break in the future. Keeping your program compatible is your responsibility as well as Apple's.

If you are unsure about a particular programming technique or feel that you must break a rule to accomplish your goals, contact Macintosh Developer Technical Support at the address listed in Technical Note #0 to see if there is another solution to your problem or a sanctioned way of working around a particular rule. If you don't ask, you will never know.

General Rules

The following are some general rules that you should follow to make your program more robust:

- Always code defensively. Check the error code after you make a call to the Toolbox. Make sure your handles and pointers are not `NIL`. Do not assume that calls will always succeed. See M.OV.CompatibilityWhy for more information.
- Use `_SysEnvirons` (and, if absolutely necessary, `HwCfgFlags`) to determine your system's configuration. When checking for the processor type, make allowances for newer Motorola processors like the 68030. See M.OV.GestaltSysenvirons for more information on `_SysEnvirons`.
- Do not check to see if MultiFinder is active (you cannot tell anyway); your application should work properly with and without MultiFinder.
- Do not make assumptions about the maximum size of a piece of memory or a resource. Do not assume that the maximum distance between two objects in memory is less than 2^{32} bytes. Do not store less than 32 bits for the size of an object (e.g., PICTs) in your data structures unless you create them.
- Call `_NGetTrapAddress` for any traps you use that are not available under A/UX (i.e., SCSI Manager traps). For a complete list of traps available under A/UX, see *A/UX Toolbox: Macintosh ROM Interface*.
- Always use the latest version of your development system and documentation. Even though you do everything "correctly," earlier versions may have bugs or inaccuracies that could break your program.

To summarize: don't make any assumptions, even if those assumptions are currently true. The future **will** change.

Hardware & CPUs

- Do not assume that you are running in the processor's supervisor mode. Do not use `TRAP` instructions or exception vectors that are reserved for future use by Apple or Motorola. See the Compatibility Guidelines chapter of *Inside Macintosh*, Volume V-1 for more information.
- Never try to bypass existing interfaces to hardware devices. Direct hardware access is not available under A/UX. Use the Serial Driver to talk to the SCC. Use the File Manager to manipulate disks. Use the SCSI Manager to talk to your non-disk devices like scanners and printers. Use QuickDraw to draw to your screen.
- Do not use timing loops. Different CPUs execute them at different speeds.

Memory Manager

Memory Manager abuse is the leading cause of death under A/UX. Here are some crucial points to remember:

- Do not set bits in master pointers directly. Use Memory Manager traps (e.g., `_HLock`, `_HGetState`, and `_HSetState`) instead.
- Do not use fake handles under any circumstances. See M.O.CompatibilityWhy for more information on `Handle` etiquette.
- When you compare master pointers, use `_StripAddress` to convert them to the correct format. See the OS Utilities chapter of *Inside Macintosh*, Volume V and M.TE.StripAddress for more information.

- Do not make assumptions about the contents of Memory Manager data structures, including master pointers and zone headers. These structures have changed under A/UX, and they will change again in the future.

Resource Manager

Here are some guidelines for using the Resource Manager:

- Avoid opening resource files read-only unless the resource file is on an AppleShare volume. If another application (including DAs and other types of code) opens (or already has open) the resource file for writing, you could end up with a corrupted resource map. If you do open a resource file read-only, you should load the resources you need into memory immediately.
- Do not set resource attribute bits directly. Use the supplied `_GetResAttrs` and `_SetResAttrs` traps.
- Do not make assumptions about the contents of Resource Manager data structures and, especially, the resource map. Do not try to walk the resource map.

WDEFs and CDEFs

In earlier versions of the System Software, the Window Manager and the Control Manager both stored the variant code (which defines how the window or control looks) in the high byte of the `defProc` field. You should use the `_GetWVariant` and `_GetCVariant` traps to get the variant code. If writing your own WDEF or CDEF, you should use the variant parameter that is passed to you.

If you are writing your own CDEF, you have to be very careful. Prior to System 7.0, there was no way to make a CDEF fully 32-bit clean, since the `calcCRgns` message uses the high bit of the region handle as a flag. *Inside Macintosh*, Volume I-331 says to "clear the high byte (not just the high bit) of the region handle before attempting to update the region." **This is wrong.** You should clear just the high bit, or your code will not run under A/UX or future versions of the Macintosh OS.

The Control Manager in System 7.0 or later has a new mechanism for calculating control regions. The following two new messages have been defined:

```
CONST
       calcCntlRgn = 10;
       calcThumbRgn = 11;
```

Whenever the Control Manager used to call your CDEF with the `calcCRgns` message, it will now do the following:

```
       IF we are in 32-bit mode
              rgnHandle := Call CDEF with message calcCntlRgn OR calcThumbRgn, as
       appropriate
       ELSE
```

```
call CDEF with old calcCRgns method, just like the good old days
```

Old CDEFs will continue to run in 24-bit mode, but they will look funny in 32-bit mode (however, they will not crash).

If your program has custom CDEFs, you should update them to support the new messages (along with the old ones) as soon as possible. Supporting the new messages will allow them to work correctly today, and in the future.

File System

Avoid building path names into your application. A/UX uses the slash (/) as a pathname separator instead of the colon (:), and external file systems implemented by Apple and by third parties may have other restrictions. For the same reason, avoid hard-coding volume and file names in your program. Try to avoid creating your own working directories. Do not assume any particular maximum length for file names; A/UX limits them to 14 characters while the Macintosh OS limits them to 31.

The easiest way to avoid intimate knowledge of the file system is to let `_SFPutFile` and `_SFGetFile` manage file names for you. Refer to *Inside Macintosh*, Volume I, The Standard File Package for details on these two calls.

Low-Memory Globals

A/UX does not support all of the low-memory globals, and future systems may provide even less support. Unfortunately, there are some things you just cannot do on a Macintosh without using low-memory globals, so it is currently impossible to avoid them entirely. Here are some guidelines:

- Avoid writing to or reading from low-memory globals unless absolutely necessary.
- Do not use low-memory globals that are labeled private, reserved for future use, or that are undocumented.
- Do not use low-memory globals when there is a trap or library routine which accomplishes the same task. For example, A/UX does not have an event queue that you can access in low memory, but it does support all of the Event Manager traps for accessing the Event Queue (e.g., `_GetNextEvent`, `_WaitNextEvent`, `_EventAvail`).

Trap Patches

Patching traps is one of the easiest ways to break your program. It is **very** difficult to write a trap patch that does not make incorrect assumptions about the way things work. Many current applications patch traps unnecessarily. If a trap does not work the way you want, implement your own code instead of trying to patch the required functionality into the trap. Here are a few guidelines to follow if you absolutely must patch a trap:

- Do not assume that `A5` is valid when you call the patch.

- Do not bypass the Trap Dispatcher to call traps directly. The performance gains are small, and there may be serious side effects.

- Do not use the Memory Manager if the trap that you are patching is not listed in "Appendix A: Routines That Move Or Purge Memory" of *Inside Macintosh X–Ref*.

Make sure that any patch you do write is not a tail patch. A tail patch is a patch which looks at the results returned by the original patch and modifies them to suit its own purposes. If you call the original trap routine with a `JSR` instead of a `JMP`, you have created a tail patch.

You need to avoid tail patches because many of Apple's System Software patches check the return address on the stack to see who called them. If you write a tail patch, you defeat these checks and may cause things to break in strange and less than wonderful ways.

Sound

If your program needs to make sounds, use the Sound Manager. A/UX (and the good old Macintosh XL/Lisa) does not support sound (yet), so your program should avoid relying on it if maximum compatibility is deemed desirable—we think it is.

Further Reference:

- *Inside Macintosh*, Volume V-1, Compatibility Guidelines
- *Inside Macintosh X–Ref*
- M.OV.Compatibility
- M.OV.CompatibilityWhy
- M.OV.GestaltSysenvirons
- M.ME.StripAddress