

New Technical Notes

Macintosh



®

Developer Support

Segment Loader Limitations

Operating System

M.OS.SegmentLoader

Written by: Andrew Shebanow

December 1988

This Technical Note discusses the jump table limitations of the Segment Loader and suggests some ways to work around these limitations to minimize the problem. These limitations are most evident to developers using MacApp and other object-oriented environments.

As application sizes increase, more and more developers are hitting a little known limit in the Macintosh run-time architecture: the Segment Loader's limit on the number of jump table entries.

The Segment Loader reads the jump table from 'CODE' resource 0. This resource has the following format:

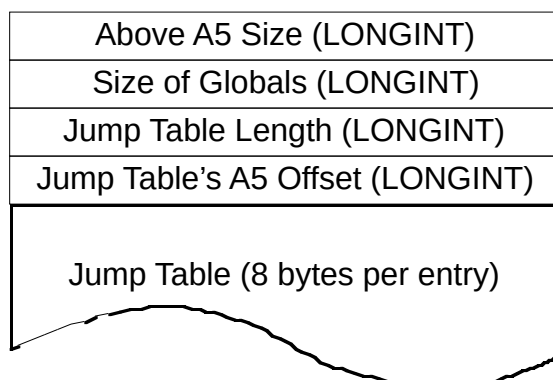


Figure 1—Jump Table in 'CODE' Resource 0

Since the 68000 uses signed 16-bit offsets for A5-relative instructions, the maximum offset that can be specified is 32767. Since each jump table entry takes up 8 bytes, there can be 4096 jump table entries, but we need to subtract the 16 bytes used by the jump table header, so the actual limit on jump table entries is **4094**.

This limit may seem rather small, but you have to remember that jump table entries are only needed if a function is called from outside of its own segment (a.k.a. an intersegment function call). For routines that are local to a segment (intrasegment function calls), the linker can call the routine without going through the jump table.

If you are developing a large program, and you think you might be in danger of running out

of jump table space, here are some guidelines you can use when designing and organizing your code:

- Segment your source code based on your run-time call chain, rather than on a simple file-by-file basis.
- Keep utility routines in the same segment as the routines which call them, rather than keeping them in a separate “Utilities” segment.
- Try to keep your segments close to the maximum size. The fewer segments you have, the fewer intersegment references you will have.

The 4094 routine limit becomes a much more significant problem when you start working with MacApp and Object Pascal. If you compile your MacApp program with debugging turned on, you will get a jump table entry for **every** method in your program. All of a sudden, an application of medium size can have jump table problems. Working around this problem is extremely difficult, since the only options available to you are:

- Giving up the MacApp debugger facilities, and using your favorite low-level debugger instead.
- Reorganizing your classes, and reducing the number of methods in your program to a reasonable level. This means that you may have to make major source code changes, and you end up with a less “object-oriented” program than you started out with.

Both of these options are extremely distasteful, but the final version of MacApp 2.0 will alleviate the problem by letting you compile your programs with both debugging and optimization turned on. The optimizer will drastically reduce the number of jump table entries, so the problem should only occur with very large programs.