A Software Developer's Guide to Switcher

Andy Hertzfeld 9 Apr 85

The Macintosh Application Switcher creates a dynamic new software environment for the 512K (or larger) Macintosh, allowing multiple application programs to reside simultaneously in memory and providing a way to switch between them very quickly. The Switcher tries to support every Macintosh application, but the interface between a program and its environment is very complex, and Switcher necessarily disturbs this delicate balance, so some programs are not compatible with it. This document is intended for an audience of Macintosh software developers and will provide suggestions and hints that will help your application to get the most out of the Switcher environment.

The most recent version of the Switcher is version 2.0, the beta release. Included with the release are new versions of the Macsbug debugger that are compatible with the Switcher. Earlier versions of Macsbug had a bug that made them not work with Switcher, so be sure to use these new versions when testing your applications out with Switcher. When you find that your application doesn't work properly with Switcher, please try to pinpoint the difficulty as precisely as possible when reporting the bug; bugs can be reported to Apple, on MAUG/Compuserve (my number is 70167,3430) or directly to me at (415) 322-8696.

One source of developer confusion is caused by the way the Switcher fudges memory size statistics. It turns out you have to allocate a 96K partition for an application to have approximately the same amount of memory it has on a 128K Mac. To avoid confusion for the typical user, the Switcher adds 32K to the size of a partition before displaying it. Thus partition sizes are "normalized" to the well-known 128K Mac; for example, a partition that's displayed as 256K actually only has 224K allocated to it.

The Switcher knows how much memory to allocate to a given program by inspecting the "SIZE -1" resource attached to the program file. The Switcher is capable of generating its own SIZE blocks using the Configure command, but it would be very nice if new applications could come "pre-configured" for their own unique memory requirements and other properties. The size block is 10 bytes long; there is a flags word followed by 2 long integers. The first long integer is the recommended size of the partition, followed by the minimum size. The values are 32K less than the virtual partition size (i.e. 96K for a 128K partition).

Currently, only the high two bits of the flags word are defined. Bit 15 means "save screen" and bit 14 means "suspend/resume" events. (see below). Unassigned bits should be kept 0 for future compatibility.

While working at Apple, I had the opportunity to watch a number of fairly complicated Macintosh applications go through their final debugging cycles. The most time-consuming and difficult part of this process is what you might call "memory tuning", which is dealing with various out-of-memory situations and with memory fragmentation. Typically, an application is tested and tuned to run on both 128K and 512K Macintoshes. However, the Switcher environment supports variable-sized partitions, so many more memory situations become possible (i.e., a 256K Mac). The biggest problem most applications have working smoothly with the Switcher is that they were tuned for 128K or 512K, and sometimes are freaked out by something in between, because they make decisions like "it's not 128 so it must be 512". Switcher-friendly applications should test memory to see how much is available and be able to deal with a wide range of memory sizes. The best way to size memory initially is to grow the heapZone out to its maximum size by requesting an enormous block, and then execute a "FreeMem" or "MaxMem" call. After initialization, the best way to determine if a certain amount of memory is currently available is to use the "ReserveMem" call. Switcher also sets up the low-memory location "MemTop" with the size of the current partition normalized to a 128K Mac, but it is not recommeded that memory-sizing decisions be based on that. Make sure you never make any assumptions about how much memory is available by looking at absolute addresses, as you don't know where you are going to be loaded.

Many applications have a need to create temporary disk files using a filename generated by the application like "Edit.Scratch" or "Paint1". Since the Switcher environment supports running the same application twice, file name conflicts are possible. A Switcher-friendly application should never used a hard-wired name for a temp file; instead, it should make up one using a random number or the time of day clock, so the names will not conflict if its running concurrently in two different partitions. For example, instead of using "Paint1", use "Paint03:12:35".

Switcher switches contexts only when an application executes a "GetNextEvent" call, so applications never have to fear being suspended when they are engaged in some time-critical activity. To be compatible with Switcher, your application must call GetNextEvent periodically, as

most applications do. To be able to cut and paste under Switcher, your application must support desk accessories by maintaining an Apple menu, and supporting cutting and pasting with desk accesories. This is because Switcher fools your application into coercing the clipBoard into global format by making think its cutting or pasting into a desk accessory. Watch out for an entry in the desk accessory menu that isn't really a desk accessory. See the discussion below on suspend/resume events for further information on the "desk-accessory charade" .

To make most effective use of memory, the Switcher supports an option where it will not save the bits of an application's screen when it switches. If this option is in effect, the program must be able to respond to update events to regenerate the screen. Thus, switcher-friendly programs should be prepared to handle update events.

Another class of Switcher incompatibilities is caused by asynchronous I/O. It is possible for a suspended application to receive control via a completion routine after its been switched out . Completion routines must be very careful about referencing low memory and globals, as they might have been swapped out and A5 is probably different. One technique for performing Switcher-friendly async I/O is to pass A5 at the end of the I/O parameter block, so your completion routine (which is passed the parameter block) can reference globals. This is only relevant to non-file I/O, as the Switcher refuses to switch if the file system is busy. Also, the Switcher suspends any vertical retrace tasks executing in a given partition when the main application is suspended, so your vertical retrace tasks don't have to worry about this.

Another potential danger area for Switcher compatibility involves timing. In the Switcher environment, it is possible for an application to be suspended for an indefinite period. Some applications use "Ticks" for relative timing; they must be careful to use 32-bit compares and arithmetic as Switcher makes it possible for more than 64K ticks to have elapsed since the last time you looked at it.

Even though the Switcher is intended to work "behind the back" of most applications, it includes some features that allow newly written programs to perform very smoothly with it. For example, if an application knew it was about to be suspended, it could clean up its act by killing I/O tasks, etc. To deal with this, the Switcher provides optional "suspend/resume" events. A suspend event means that the

next time you call GetNextEvent, you will be suspended. A resume event is the first event you get back after you've been re-activated following a suspension. Suspend and resume events are both reported as event 15 (formerly an application-defined event). The high byte of the message field is set to 01 to indicate that its a suspend/resume event (eventually event 15 will be used for other purposes as well). The lowest bit of the message field (bit 0) is clear if its a suspend event and set if its a resume event. The next bit up (bit 1) is set if clipboard coercion is required. The Switcher uses bit 14 of the flags word in the SIZE record to indicate if a program should receive suspend/resume events. If this bit is set, the Switcher won't put on the desk accessory charade for clipboard coercion, as it assumes that the application is converting the clipboard when requested to in the suspend/resume event.

An even more exciting area is that of "interlocking" applications that run fine by themselves but are ultra-integrated when run together under the Switcher. Applications can find out the global picture by inspecting Switcher globals. Switcher globals are accessed through a low-memory pointer kept at address $282. If the pointer kept there is -1 or 0, it means that the application is not currently running under the Switcher. Otherwise, its a pointer to a public table of Switcher global variables. The first 8 longWords in the table is a list of pointers to the base of all currently active applications, or zero if no application is present in a given slot. By inspecting this table, applications can determine what other applications they are coexisting with and where they are located. Apple will eventually provide complete documentation on effectively using this "world" table, as well as detailing some other useful Switcher globals.

All in all, it is amazing to me how many programs do run properly under the Switcher. Most developers should not have to worry or understand all of the stuff discussed here; ordinary applications usually run just fine. It has been fun working on the Switcher. To diagnose various crashes, I had to trace through the guts of lots of very different applications. Countless features and strategy shifts were implemented to support this or that application. As a designer of the Mac system, I'm one of the very few who never had the experience of trying to learn how it works. In a way, developing the Switcher was the application programmer's revenge for that experience. It certainly gave me a new appreciation for our strange and wonderful software base.