Inside Switcher

by Andy Hertzfeld

Summer 1985
(First Draft)

Introduction

Switcher is a small but powerful utility program (about 16K of tight assembly language) that creates a dynamic new software environment for the 512K (or larger) Macintosh and Macintosh XL systems, allowing multiple application programs to reside simultaneously in memory and providing an easy way to switch between them very quickly. Unlike the Switcher user manual, this document describes Switcher from the inside out, and is intended for a rather sophisticated audience of software developers (you know, the Inside Mac crowd). It explains how Switcher works and provides hints for customizing its behavior, offering techniques that allow your application to get the most from the Switcher environment.

Switcher has been distributed in many different versions as it went through its testing stages. The current version of Switcher is 4.4, the first production release. If you are having a problem with Switcher, make sure you are running version 4.4 by reading the version number displayed in the "About Switcher" dialog. You can obtain the latest release through Apple Technical Support, to which you can direct your questions and bug reports.

Theory of Operation

Switcher performs some of the same functions as the Finder and the loader, allowing users to select and launch applications and documents. Instead of using the ROM "Launch" routine like the Finder does, Switcher uses a special routine that launches an application into a subset of available memory, allowing multiple applications to reside simultaneously in memory. An application's current state (all memory locations and register values that are specific to an application) is mostly concentrated in its heap and stack areas but also includes various low memory globals and system heap objects. Switcher must maintain multiple copies of these shared low memory and system heap areas and save and restore them during context switches. To switch between applications, Switcher saves these special memory areas into a **process state record** associated with the current application, and then replaces them with values belonging to a new application.

Since the Macintosh OS was not designed to support concurrent applications, Switcher must modify a number of system ROM routines using RAM-based patches. Currently, Switcher modifies or replaces 26 different system routines. The most important of these is the "GetNextEvent" routine, which is called periodically by most law-abiding applications. Besides performing the usual event-handling functions, Switcher's GetNextEvent routine checks to see if the user is requesting a context switch by polling the mouse position and the keyboard bitmap. If so, it suspends the current application and switches control to a new one.

The Macintosh clipboard provides a flexible facility for transferring data between different applications. Since there is often some overhead associated with writing to the clipboard, most applications don't put stuff into it every time a cut or paste occurs. However, they must make sure the clipboard is valid to cut and paste with desk accessories. Thus Switcher has a devious way to request applications to convert their clipboard to global format -- it makes them think that they're cutting or pasting into a desk accessory. When the user requests a switch with clipboard conversion, the Switcher passes a sequence of phony events to the application to make it think it's pasting into a desk accessory, starting with a mouse-down in the menu bar area to get it to think it's pulling down the desk menu. This procedure is highly heuristic and evolved by inspecting the behavior of dozens of applications (see below for a full description). The bottom line is that your application must support cutting and pasting with desk accessories to be able to cut and paste with other applications under the Switcher.

How to determine if you're running under Switcher

Low memory location $282 is called SwitcherGlobals and it may be inspected to determine if your application is currently running under Switcher or not. If the longword stored there is 0 or -1, you are not running under Switcher; otherwise the longword is a pointer to the Switcher globals area (discussed below).

If you have code running at the interrupt level (like a completion routine), it is sometimes necessary to determine if your application is currently the active one. This is easily done by inspecting a unique low memory location like ApplZone ($2AA) when your application is initialized, and then comparing that saved value with the current value; if they're the same, you are currently active. Note that your can't store the value in your global area as you can't count on A5 being right; nor can you store it in the application low memory area. One useful technique is to pass A5 at the end of your I/O parameter block (or VBL queue element) so you can address your globals and therefore store it there; of course you can just store it in reserved space in your code segment. Another reliable way to tell if you are currently active without accessing your global is to compare the theTask handle in the Switcher globals area (see below) with the process state block handle in your world table entry, accessing via your world table entry (see below again).

The World Table

As mentioned above, the low memory location known as "SwitcherGlobals" (address $282) contains a longword that is a pointer to the Switcher global area (or 0 or -1 if you're not running under Switcher). That pointer points to a very important Switcher data structure called the **world table** that allows an application to determine what other applications it is currently running with and provides access to other application's heapZone and low memory areas.

The world table consists of the first 8 longwords of the switcher global area. Each represents an individual switcher slot. If the value of the longword is zero, that slot is empty, otherwise it is a pointer to a switcher "world".

A switcher "world " is a large continguous chunk of memory that an application runs in, that is a non-relocatable block allocated in the Switcher's heapZone. It consists of an 18 byte header containing information maintained by Switcher, followed by the start of the application heap zone of the world. The format of the world header is as follows:

offset 0:	handle to the process control block
offset 4:	flags word
offset 6:	address of background process routine (0 if none)
offset 10:	handle to saved screenbits (0 if none)
offset 14:	windowPtr for phony desk accessory (used for clipboard coercion)
offset 18:	start of application heap zone

The process control block handle is a handle to the data structure that contains the current state of a suspended application, which is described in detail in the next section. The flags word is used to keep track of some of the properties of the application. If bit 15 is set, the screen handle kept at offset 10 (if any) is currently valid. If bit 14 is set, the application should be sent suspend and resume events it does not need to be subjected to the "desk accessory charade" to convert its clipboard. Bit 8 is used to flag if an application is inactive with it's clipboard converted; it is set if that's currently the case. Bit 0 is set when an application is running in the partition (even if it's currently suspended).

The next field in the world header is the background task routine at offset 6 If the value stored there is zero, the world has no associated background task; otherwise it contains the address of a routine that is called in a round-robin fashion with other similar routines when the system is idle (see below). Offset 10 contains the handle to a large block of memory containing the saved contents of the current screen bitmap, which is zero if no screen is being saved. Bit 15 of the flags word must be set or the handle does not currently contain valid data. If an inactive application wishes to write something onto its screen, it should set the port's bits to the address of this handle dereferenced. The field at offset 14 is used for the internal operation of Switcher during clipboard conversion and will not be described here.

The application heap zone of a given world starts at offset 18 past the world pointer. Perform a SetZone to that address to manipulate the heapZone of an inactive application (see below).

The Process Control Block

Every application running under Switcher has a process control block associated with it that contains various low memory values, system heap handles and other information that comprise the state of the system that must be multiplexed between all the currently executing applications. The first entry in the world header is a handle to the process control block belonging to that world.

The process control block consists of nine different parts. It begins with a longword that indicates the current size of the process control block, including itself. Next comes the saved low memory, at specific offsets (described below) so it may be accessed by other programs when an application is suspended. That is followed by the saved dispatch table (if MemToSwitch is NIL), or a variable amount of additional memory as described by the MemToSwitch table (see below for a description of the MemToSwitch data structure). That is followed by information necessary to restore any suspend vertical retrace task (a VBL receiver address and longword tick count for each suspended element, terminated by a longword of 0). Next comes a list of handles in the application heap that must be patched in the system heap map. Since resources may be added to the system file while Switcher is running, these are identified by a logical address (type and ID), rather than a physical offset. Next comes a list of handles that must be patched in the unit table, as desk accessories and drivers can live in the application heap and therefore are part of the state of the application. After that comes a copy of the application parameter handle passed in by the Finder, which is another system heap entity that must be multiplexed between the applications. Next comes a descriptor of the state of the sound driver and hardware, so that different applications can share the sound driver. Finally, there is the address of the stack pointer for the application at the time of its suspension. The remaining registers and program counter are saved on the stack prior to suspension.

Defining Switchable Memory

One of the design goals of Switcher was to include the ability for it to work properly in future Macintosh software environments, such as the one implemented by the 128K ROM. Future environments may define memory locations that are part of an application's state and therefore must be switched. To solve this problem, Switcher inspects a low memory location called MemToSwitch (location $286), which contains a pointer to a data structure that may be set up by a future software environment. If its value is -1 or 0, Switcher assumes that it's operating in the normal environment defined by the initial

Macintosh ROM. Otherwise, it interprets the value as a pointer to a data structure defining additional memory to include as part of an application's state.

The data structure is quite simple. It consists of pairs of words, the first defining the address of a block of sequential memory, while the second word contains the length (in bytes) of the block. An address of 0 must follow the last pair to terminate the data structure. The address is kept to a word since most switchable areas are in low memory and it therefore saves space, but a provision is made for longword addresses. If the high bit of the address is set (i.e., the address is greater or equal to $8000), it considers it to be the high word of a longword that is the one's complement (i.e., NOT.L) of the desired address. The size is still kept as a word value.

Since there might be multiple software environments running simultaneously, the cautious programmer will not simply store a pointer in the MemToSwitch location but rather will inspect it to see if the data structure already exists (i.e., MemToSwitch is not -1 or 0). If it already exists, you should copy the information there into your own block so enviroments may be nested.

Since new operating environments are likely to change the location and size of the line 1010 dispatching tables (currently 1K long at location $400) Switcher assumes the location of the dispatching tables will be included in the MemToSwitch data structure, even if it hasn't changed from the nominal locations.

Normal applications do not have to worry about the MemToSwitch data structure; it is only for system programmers that are creating new environments that wish to work with Switcher.

Accessing the low memory of inactive applications

It is sometimes useful for one application to be able to access the low memory of another suspended application. This may be done by examining fixed offsets in the process control block.

The following table describes which offsets in the process control block correspond to which low memory values:

Offset 4:	MonkeyLives	($100)
Offset 6:	MemTop	($108)
Offset 10:	BufPtr	($10C)
Offset 14:	StkLowPt	($110)
Offset 18:	HeapEnd	($114)
Offset 22:	theZone	($118)

Offset 26:	ApplLimit	($130)	
Offset 30:	SEvtEnable	($15C..$15F)	(4 bytes long)
Offset 34:	ApplZone	($2AA)	
Offset 38:	MinStack	($31E..$33F)	(34 bytes long)
Offset 72:	GrafBegin	($800..$AFB)	(764 bytes long)
Offset 836:	DefVCBPtr	($352)	(4 bytes long)

Note that the entire range of graphics, toolbox and loader globals are saved consecutively beginning at offset 64. Thus to access the WindowList of a suspended application (which is normally at low memory address $9D6) I would use offset 542 ($9D6-$800+72). To find out the name of a suspended application, which is stored at $910, I would use offset 344 ($910-$800+72).

Note also that the contents of the process control block are valid only when an application is currently suspended. If an application is active, look in actual low memory for the current values. If an application has never been suspended, the state of its process control block is undefined.

Accessing other HeapZones

When developing a series of highly integrated applications, it is sometimes useful to read and write to a heapzone of another application. It is simple to read information from another heapZone -- you can access the low memory of a suspended application in the manner described above to pick up a pointer or handle to information in the suspended application's heapzone. It is harder, but possible, to allocate data in a suspended application's heapzone. To do this, you must save a few critical low memory locations on the stack: location $114 (heapEnd) and the twelve bytes at location $328 (growZone information). Replace these locations with information from the suspended application's process control block, as described above. Then perform at SetZone to the address of the application's world base + 18 (or offset 34 in the process control block, which should be the same). Now you can make various memory manager calls. After you're finished, resave the low memory locations back into the process control block, as they might have changed. Finally, replace the low memory locations with their original values saved on the stack.

Any application whose heapZone has memory allocated in it while it's suspended must take care in it's GrowZone procedure not to access any low memory or global variables, as low memory and A5 may not be properly set up at the time GrowZone is invoked. If interlocking applications are properly architected, it is very rare to have to allocate memory in the heapZones of suspended applications; the applications should allocate a spare block of appropriate size before they are suspended, and leave a pointer or handle to it in the ApplScratch low memory area, thus avoiding the necessity of allocating memory when they are suspended.

Switcher Globals

The Switcher maintains a number of global variables to help it perform its various functions. It is sometimes helpful for an application to access them. As mentioned above, Switcher globals are pointed to by location $282. If the longword stored there is 0 or -1, you are not currently running under Switcher; otherwise $282 contains a pointer to the Switcher globals.

The first 32 bytes of the Switcher global area are taken up by the world table, as described above. At offset 32 of the Switcher globals is a handle called "HostTask", which contains the process control block handle for the Switcher itself. Offset 36 is called "TheTask" and contains the process control block handle for the currently active task. If HostTask equals theTask, the Switcher is the currently active application. The name of the current application made be determined by inspecting offset 344 (see above) into the process control block specified by theTask. Next, at offset 40, is the flags word describing the properties of the currently active application.

The next 8 bytes are scratch locations used for parameter passing. Next, at offset 50, is a boolean called ArrowEnable. This allows an application or desk accessory to turn off the Switcher arrow that appears in the menu bar area, so it may use that space for other purposes. If the byte stored at ArrowEnable is 0, the Switcher arrows will not be displayed and no mouse hit-testing will be performed for them. Anyone who changes the state of the ArrowEnable boolean is also responsible for updating the menu bar graphics to reflect the change. This means calling "DrawMenuBar" for the active application, and invalidating the saved display by resetting bit 15 of the flags world in the world table (see above) for any inactive application.

The next byte, at offset 51, is called ClipConvert and is a boolean maintained by the Switcher application that contains the state of the "Always Convert Clipboard" flag. The next word, at offset 52, is called Hibernation. It's normal state is 0; it is non-zero only when the Switcher is exited with one remaining application. That application is switched to and the Switcher goes into "hibernation" by setting that boolean.

Next, at offset 54, is a longword called MainZone, that contains a pointer to the Switcher's heapZone, which is the meta-zone that all the other partitions are allocated in. The next 6 longwords, beginning at offset 58, is an address table holding the addresses of selected Switcher core routines so they can be accessed by an application or desk accessory. See the section below on "Switching under program control" for a description of these routines.

Only two other globals are of interest. Offset 90 is called NextTask and it contains an integer which specifies which world is next one to be switched into. Finally, offset 96 contains the string that represents the name of the Switcher application. An application may transfer control back to the Switcher by calling OpenDeskAcc with that name as the parameter.

Finding out which other applications are running

Using the information described above, it should be easy to find out the names of all the programs we're running with. Using the switcher globals pointer (at $282) to gain access to the world table, you inspect the 8 longword pointers in the world table. The first field in the world data structure is the process control block handle. At offset 344 into there is a string containing the name of the application.

Here is some sample code to illustrate the technique:

```
            MOVE.L      SwitcherData,A3     ;get ptr  to world table
            MOVEQ       #7,D3               ;8 worlds to inspect
WorldLoop
            MOVE.L      (A3)+,D0            ;get world pointer
            BEQ.S       NextWorld           ;if slot is empty, skip

            MOVE.L      (A0),A0             ;get pcb handle
            MOVE.L      (A0),A0             ;handle -> pointer

            PEA         344(A0)             ;push ptr to application name
            _DrawString                     ;just to do something with it
NextWorld
            DBRA    D3,WorldLoop            ;loop through 8 worlds
```

Switching under program control

Suspend/Resume Events

Applications can optionally receive suspend/resume events. If an application wants to receive them, it should include a SIZE -1resource that has bit 14 set in the flags word (see below; offset 0); otherwise this bit should be clear (or the application could have no SIZE resource at all; they default off). 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. Additionally, if a program is receiving suspend and resume events, 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.

Background Processing

There are many time consuming tasks we often perform that don't require the full attention of the Macintosh or ourselves; we'd like to be able to do some other useful work while they execute "in the background". Some common examples are file downloading and printing. Switcher supports a mechanism by which applications can arrange to get called even when they are currently suspended, thereby allowing them to execute in the background.

Offset 6 in the world table for each application is the address of a routine that Switcher will call every once in a while when it sees a null event come through. The Switcher goes through all currently installed background task handlers in a round robin fashion, call one each null event. An application can access its world table entry by taking the pointer at ApplZone ($2AA) and subtracting 18 (see above). It installs a background routine by storing a non-zero value at offset 6 in the world table entry; it removes a background routine by storing zero there.

At the time a background task is invoked, its world table entry pointer is passed to it in A1. From there is can pick up the process control block and access its saved low memory if neccessary. In particular, it can pick up its current A5 value at offset 332 in the PCB so it can access its globals as necessary. If it is necessary to allocate information in the heapZone, it can employ the techniques described in "accessing other heapzones" (described above); this should be avoided if possible and often can be, by pre-allocating memory before the application is suspended.

If installed, the background routine of an application will be called even when the application that owns it is currently active. Sometimes this is desirable while in other circumstances it is not. It can be avoided by installing the background task in the world table when you get a suspend event, and zeroing it upon receiving a resume event. Another technique is to always leave it installed, but check theTask (accessed via SwitcherData) with the world table entry (accessed via A1) and doing no work if they match.

Vertical Retrace Tasks and Asynchronous I/O

Configuring your application

Switcher is capable of allocating variable amounts of memory to each application. It is very tough for an end user (or even a developer!) to figure out the optimum amount of memory for a given program. The Switcher provides a mechanism that allows a program to specify its preferred memory configuration, as well as some other properties. It discovers this information by inspecting the "SIZE -1" resource attached to the program's resource 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, and its format is as follows:

Offset 0:	Flags Word
Offset 2:	preferred memory size
Offset 6:	minimum memory size

There is a flags word followed by 2 long integers. The first long integer is the preferred 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 above). Unassigned bits should be kept 0 for future compatibility. If there is no size resource attached to an application, the default configuration of < screen size on, suspend/resume off, 128K prefered and 128K minimum > is used.

Configuring Switcher

There are a number of ways that Switcher itself can be configured by changing some of its resources. Of course all text used by Switcher is kept in resources, so it may be translated into foreign languages with having access to its source code.

The resource ESCK 256 is a 6 byte resource that contains the key-codes and ASCII for the keys that control switching. The default value of ESCK 256 is "001E212A5D5B" (in hex). The bytes are interpreted as follows:

Offset 0:	ignored; should be 0
Offset 1:	the keycode for "switch right"
Offset 2:	the keycode for "switch left"
Offset 3:	the keycode for "switch back"
Offset 4:	the ASCII for "switch right"
Offset 5:	the ASCII for "switch back"

The resource CFIG 0 is an 8 bytes resource that contains the state of the Switcher options, as set by the options dialog. The default value of CFIG 0 is "0000 0000 FFFF 0000" (in hex) The format of the 8 configuration bytes is interpreted as follows:

Offset 0:	ignored; should be 0
Offset 1:	Disable Keyboard Switching
Offset 2:	Reverse Switch Directions
Offset 3:	Same One Twice
Offset 4:	Back After Launch
Offset 5:	Switching Animation
Offset 6:	Always Convert Clipboard

Offset 7: Switcher in Rotation

When the Switcher is launched, it scans the volume it is on for a Switcher document of a certain title. If one is found, that document is automatically opened. The name it uses for the start-up document is kept in the resource STR 40. It current defaults to "Switcher.StartUp".

Being Switcher Friendly

Converting the ClipBoard

Re-entrant Applications (Same One Twice)

Switcher Document Format