About MultiFinder

In general, if you write an application using the traditional Mac programming model, you will be perfectly compatible with MultiFinder. This topic summarizes the few items to be aware of when you want to take advantage of the MultiFinder environment.

Task Switching

MultiFinder will suspend your application only when you make an "event call" (**GetNextEvent**, **WaitNextEvent**, or **EventAvail**). If your application is MultiFinder-aware (as indicated by a 'SIZE' resource), it will receive a "suspend" event and can take some action to prepare to be switched (though most often there is nothing you need to do). MultiFinder **will not** switch you out if your front window is a dBoxProc window (i.e., while you are executing a modal dialog).

There are three classes of context switch in MultiFinder. In a major switch, your application is sent from the foreground to the background. The A5 world is switched, and so is your low-memory world. If you accept the Suspend and Resume events that MultiFinder sends, your application will know when it is sent to the background.

In a minor switch, no Suspend or Resume events are generated, and the front application stays front, but the A5 and low-memory worlds are switched, so background processes can do some work.

When MultiFinder detects that a background application's windows need to be updated, it will send update events to the background application, causing an update switch. This is much like any minor switch, except that your application need not be MultiFinder-aware to receive these update events.

Multi-tasking

It is possible to write code to be executed in the background (i.e., while some other application is in the front). However, MultiFinder employs "cooperative multitasking" which means that other apps must give you time every once in a while (by making a **WaitNextEvent** call). MF is not really set up to run concurrent threads of execution, as in "pre-emptive multitasking" environments like Unix, OS/2, and others. If you are working in the background, be as friendly as possible. Don't monopolize the CPU. Do your work a bit at a time, periodically making calls to **WaitNextEvent** to relinquish control.

Event Handling

A MultiFinder-aware application should call **WaitNextEvent**, rather than **GetNextEvent**. This provides the maximum amount of time for background tasks and provides a way to get notified when you get juggled. See **WaitNextEvent** for a way to see if this trap is available and a skeletal example of usage. Note that **WaitNextEvent** may be available whether MultiFinder is running or not.

There are several tricks to using **WaitNextEvent**.

- Rember to check to see if the **WaitNextEvent** trap is available before calling it. You might want to use **GetTrapAddress** and compare it to the "unimplemented trap" (0xA89F) to see if it is available or not.

- You won't receive null events as often, so you should not employ them to perform periodic maintenance, like monitoring the cursor position. You can set up a "mouse region" to watch for mouse motions and an app4Evt-type event will be generated to let you maintain the cursor. Note that app4Evt has been replaced in System 7 with OSEvt.

- You should provide a 'SIZE' resource in your application, and make provisions to watch for "suspend" and "resume" events (special variations of app4Evt events) in your main event loop. If the 'SIZE' resource's "multiFinderAware" and "acceptSuspendResumeEvents" flags are set, you must activate your front window when you receive a "resume" event, and deactivate it in response to a "suspend" event.

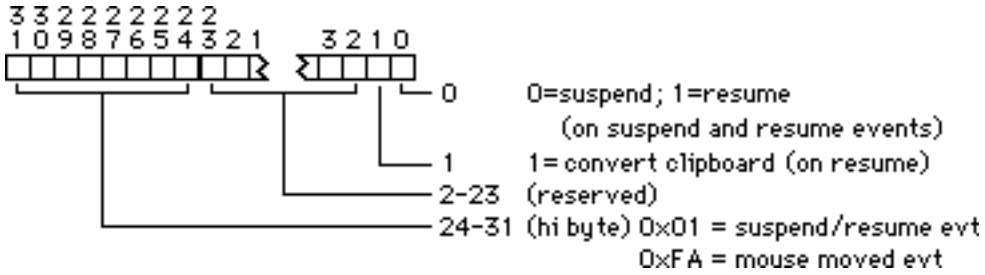
- If you use a private scrap (clipboard), you are responsible for converting and copying it to and from the desk scrap when you receive a suspend/resume event. If you already use the desk scrap for your cut and paste operations, there is no need to do anything special.

- Provide a "mouse region" so that you can keep track of the cursor. This is of particular interest if you use special cursors in certain areas of your application.

- Don't call **SystemTask**. All actions performed by this function are now incorporated into **WaitNextEvent**.

app4Evt Events

When EventRecord.what is an app4Evt event, the message field is formatted as:



Whether you actually get any of these events depends on bits of the 'SIZE' resource.

Note that the app4Evt is not used in System 7 and has been replaced by OSEvt.

Suspend/Resume: When the high byte of EventRecord.message is 0x01, this is a suspend or resume event. On suspend/resume events, check bit 0 and respond as follows:

- When message bit 0 is clear, this is a "suspend" event. Convert your scrap if needed, and use **PutScrap** to store it in the desk scrap. Deactivate your windows if the MultiFinder-aware bit in your 'SIZE' resource is set. You will be juggled out on your next event call.

- When message bit 0 is 1, this is a "resume" event. If bit 1 is clear,

you should copy the desk scrap and convert it to your private format (if needed). If the MultiFinder-aware bit is set, you must explicitly re-activate your window(s) yourself.

Mouse-Moved: If the high byte is 0xFA, this is a "mouse-moved" event. You will get these events when you specify a non-null *mouseRgn* in the call to **WaitNextEvent** and the user positions the mouse out of the *mouseRgn* area. A more direct name for this type of event would be "a mouseNotInRgn event'. As long as the mouse is not in *mouseRgn*, you will continue to receive these events.

An effective program-design strategy is to subtract the area of interest (for example, where you want to monitor the cursor shape) from the region of the desktop (GrayRgn). Then you will receive these events only when the mouse is where you want to keep an eye on it.

If you want to find out if the mouse moved, you must provide a *mouseRgn* that does not include the current mouse point. **WaitNextEvent** for an example that uses the *sleep* parameter to make sure you set the cursor back to an arrow when the mouse wanders beyond the region boundary.

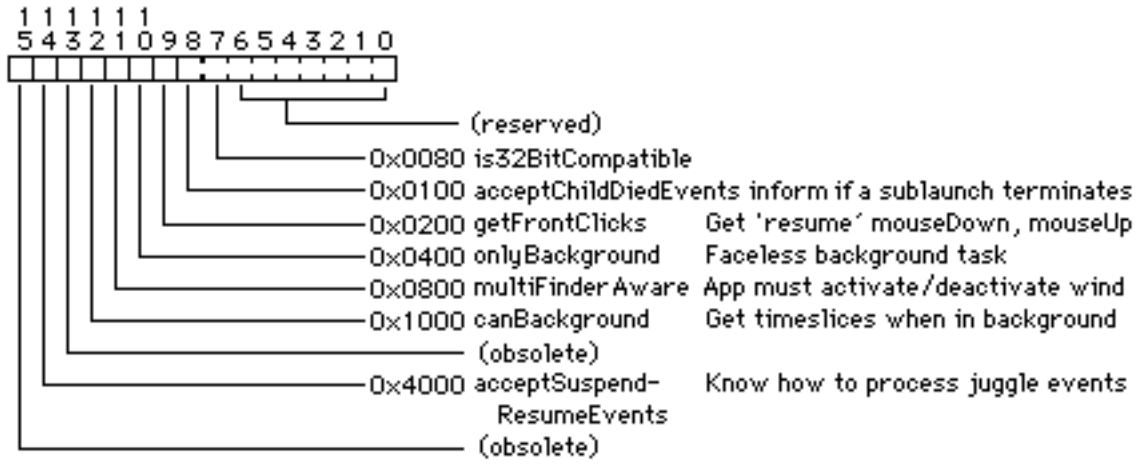
The 'SIZE' resource

MultiFinder-aware applications should have a 'SIZE' resource in their application resource. This resource identifies the memory requirements and several MultiFinder options. An application should include a 'SIZE' resource with an ID of -1. The user can change the memory setting via the Finder's "Get Info" dialog. When that's done, the modified memory setting is stored in 'SIZE' resource ID 0. When MultiFinder starts an application, it looks for the 'SIZE' 0 resource, and if unable to use it, goes to the original (-1) resource to determine the partition size. If MultiFinder doesn't find a 'SIZE' resource, it assigns a default partition of 384K.

A 'SIZE' resource is a 10-byte structure formatted as follows:

size	offset	description
(word)	0	MultiFinder flags (see below)
(long)	2	preferred size
(long)	6	minimum size
	10	(size, in bytes, of 'SIZE' resource)

You must determine the two size values empirically. Use the "About Finder" dialog to see how close your application comes to filling its partition. Prior to System 7 the bits of the MultiFinder flags were formatted as follows:



With the advent of System 7, some additional flags have been added. See **'SIZE' Resource** for a discussion of all the flags now available in the 'SIZE' resource.

Setting the "is32BitCompatible" flag means that your application is "32-bit clean", i.e., it will run in the 32-bit addressing mode of A/UX and System 7.0 and later.

If the "**acceptChildDiedEvents**" flag is set, certain development versions of MultiFinder will send your application a message to tell you that the programs your application sublaunched have terminated. Since commercially available versions of MultiFinder do not support it, this is not a flag to use in normal programming.

The "**getFrontClicks**" flag should normally be clear. When set, your application will get a mouseDown and mouseUp event when a user reactivates it by clicking on a window. For this Guide, the "**getFrontClicks**" bit is set to take notice of a double click on the iconized window (i.e., to bring itself up with the least amount of user interaction).

Setting "**canBackground**" means that you expect to get time whenever there is nothing going on in the foreground. You will get a null event and you can take a short while to do some processing, then call **WaitNextEvent** to return control to anyone who wants it.

An application that sets the "**onlyBackground**" bit must also set "canBackground'. Such applications work entirely in the background and have no user interface. You may use Notification Manager calls to get the user's attention, if absolutely necessary.

Setting the "**multiFinderAware**" bit means you take responsibility for deactivating or activating the window when you get a suspend or resume event. If you don't set this bit, you will receive activateEvt events for your window when you are juggled in or out. By setting this bit and handling window activation and deactivation, you ensure the fastest possible task switching.

The "**acceptSuspendResumeEvents**" bit indicates that you know how to process the new app4Evt events and that you will convert your private scrap to the the desk scrap as needed. Unless set, you will not receive the

suspend/resume variation of the app4Evt event. **Note that the app4Evt is not used in System 7 and has been replaced by OSEvt**.

Automatic Open and Quit

When you double click a document in the Finder and its application is already running, MultiFinder juggles the application to the foreground and simulates the events of selecting Open... from your File menu. It then short circuits your call to **SFGetFile** by passing back the information about the double-clicked document.

Likewise, when you select Shut Down from the Finder's Special menu, MultiFinder simulates the selection of your Quit menu item.

If you use a non-standard name for File, Open... or Quit, you should create a set of 'mstr' resources so MultiFinder can figure out which menu item to invoke:

'mstr' ID 100	Name of your File menu (menu containing "Quit")
'mstr' ID 101	Name of your Quit menu item
'mstr' ID 102	Name of your File menu (menu containing "Open...")
'mstr' ID 103	Name of menu item used to open documents

The format of an 'mstr' resource is identical to that of an 'STR ' resource. You may also use an 'mst#' resource (identical to an 'STR#' resource) if you want to provide a series of names for MultiFinder to try to match.

If you are running System 7, your application should handle Open Application event, the Open Documents event, the Print Documents event and the Quit Application event. For more information, see **Required Apple Events**.

Desk Accessories

Under MultiFinder, Desk Accessories are opened in a DA "layer". This means that they don't occupy space on your heap and you don't need to worry about passing DA events to them (exception: if a DA is opened while the *option* key is pressed, it is loaded and gets executed as with Finder).

Under MultiFinder, a program derives very little advantage from being a DA. Apple recommends you simply write DA-like applications as standard applications, and allow the user to juggle them as needed.

Interrupt Time

If your application uses routines that execute at interrupt time and access global data, those routines need to determine whether their A5 world is set up correctly. At the entry to **Vert. Retrace Mgr** tasks and **asynchronous I/O** completion routines, A0 is guaranteed at the entry to the task to point to a parameter block after which you can store the value of A5 prior to installing the routine. The routine can set up A5 from this stored value when it is called.

Use this same strategy for **Time Manager** tasks under System software newer than 6.0.3. For **Time Manager** tasks under older versions of the System software, and for interrupt service routines, the only way to make A5 available is to write it directly into your routine's code. Apple strongly

discourages this kind of self-modifying code, and warns that it runs a risk of incompatibility.