Macintosh Technical Notes

#205: MultiFinder Revisited: The 6.0 System Release

See also: *Programmer’s Guide to MultiFinder* (APDA)
MultiFinder Development Package (APDA)
Technical Note #117, Compatibility: How & Why
Technical Note #158, Frequently Asked MultiFinder Questions
Technical Note #177, Problem with _WaitNextEvent in MultiFinder 1.0
Technical Note #180, MultiFinder Miscellany
Technical Note #212, The Joy Of Being 32-Bit Clean

Written by: Dave Burnard August 1, 1988
Revised by: Andrew Shebanow December 1988

This Technical Note describes several new features found in MultiFinder 6.0 and a few more commonly-asked questions.
Changes since October 1, 1988: Added new `'SIZE'` resource flag bits for `32BitCompatible` and `getChildDiedEvents`.

How Can I Tell if MultiFinder is Present

Once again, you can’t. Previous Technical Notes discuss how to check for the new services available with MultiFinder (i.e., `_WaitNextEvent` and the temporary memory allocation calls).

Currently, since an application cannot tell if MultiFinder is present, the application also cannot know how a sublaunch will behave (see Technical Note #126). Unfortunately, the two possible sublaunch behaviors are radically different; with MultiFinder the `_Launch` trap will return to the application and without MultiFinder it will not. For most applications,however, these differences in sublaunch behavior should not matter. Hopefully, the `_Launch` trap will be improved in a future System Software release.

_WaitNextEvent is Always Available

In System 6.0 and later, `_WaitNextEvent` is present whether or not MultiFinder is installed. Calling `_WaitNextEvent` without MultiFinder installed is virtually identical to calling it with MultiFinder installed. Your application can still “sleep” for a specified time and be notified if the cursor location is outside a specified region. The only difference is that your application will not be suspended or resumed when MultiFinder is not installed. If your application requires System 6.0 or later, we recommend calling `_WaitNextEvent` instead of `_GetNextEvent` in your main event loop.

_MFMemTop Errata

The *Macintosh Programmer's Guide to MultiFinder*, dated June 3, 1988 and distributed through APDA, incorrectly documents `_MFMemTop` on page 90. `_MFMemTop` does not return a pointer to the top of your application's memory partition as it is documented. It does, however, return a pointer to the top of the physical RAM of your machine.

_MFTempHandles Are Not Handles

The MultiFinder temporary memory allocation call, `_MFTempNewHandle`, currently does not return a "true" `Handle` in the sense that it can be used interchangeably with a `Handle` obtained from `_NewHandle`. Specifically, you cannot pass a `Handle` obtained from `_MFTempNewHandle` to any Memory Manager routine or Toolbox routine which, in turn, passes it to the Memory Manager (either directly or indirectly). Like a "true" `Handle`, however, you can still dereference a `Handle` from `_MFTempNewHandle`. You should treat a `Handle` from `_MFTempNewHandle` in the same way you would a fake `Handle` (i.e., a `Handle` not obtained from the Memory Manager—see Technical Note #117). This restriction on the use of MultiFinder temporary memory may not apply in future System Software releases.

Mouse-Moved Event Confusion

There has been some confusion over the `mouseRgn` parameter to `_WaitNextEvent`, and under what circumstances it will return a mouse-moved event. Most of the confusion is caused by the word "moved." Many applications have assumed that mouse-moved events are generated only when the mouse actually leaves the mouse region. In System 6.0 and later, `_WaitNextEvent` will return a mouse-moved event whenever the cursor is outside the mouse region. Thus, when an application receives a mouse-moved event, it should compute a new mouse region based on the new cursor location before calling `_WaitNextEvent` again, otherwise `_WaitNextEvent` will continue to return mouse-moved events until the user moves the cursor back inside the mouse region, or a new mouse region is specified.

New MultiFinder Features—Open Document and Quit

In System 6.0 and later, MultiFinder adds the ability to open application documents from the Finder even though the owner application may already be open. For the moment, MultiFinder accomplishes this by simulating a mouse-down event in the application's menu item for opening files. The application will usually respond by calling `_SFGetFile`, which MultiFinder short circuits into returning the document opened in the Finder layer. This is similar to the way that MultiFinder triggers applications to quit when the user selects Shut Down or Restart from the Finder's Special menu.

In future System Software releases, this mechanism will probably change to a more straightforward method of notifying the application that it needs to open a document or to quit. How does MultiFinder find the Open item, you ask? By default, MultiFinder looks for a File menu with an item named Open…, Open …, Open..., etc. Of course some applications will not have a File menu or will have named their Open item something different like Open Document. To allow for this, MultiFinder will look first in the application's resource fork for resources of type `'mstr'` or `'mst#'` in the range 100–103. An `'mstr'` resource has the same format as an `'STR '` resource (just a Pascal string) and contains the name of the menu or menu item for which MultiFinder should look. An `'mst#'` resource has the same format as an `'STR#'` resource (a list of Pascal strings) and contains a set of names for the menu or menu item for which MultiFinder should look. MultiFinder uses the same mechanism to locate the application’s Quit command.

Res ID	**Meaning**
100	Name(s) of the menu containing the quit command
101	Name(s) of the menu item corresponding to the quit command
102	Name(s) of the menu containing the open document command
103	Name(s) of the menu item corresponding to the open document command

As always, be careful to avoid any "clever" tricks that rely on this information, MultiFinder will not always work this way.

New MultiFinder Features—Additions to the 'SIZE' Resource

The `'SIZE'` resource has four new flags (`onlyBackground`, `getFrontClicks`, `getChildDiedEvents`, and `32BitCompatible`) which communicate information about an application to MultiFinder. Figure 1 below illustrates the locations of these new flags. Setting both the `onlyBackground` flag and the `canBackground` flag informs MultiFinder that an application is a "faceless background task," that is, it has no user interface (i.e., no windows and no ports) and should only be run in the background. For example, the application Backgrounder is a faceless background task with both `onlyBackground` and `canBackground` set.

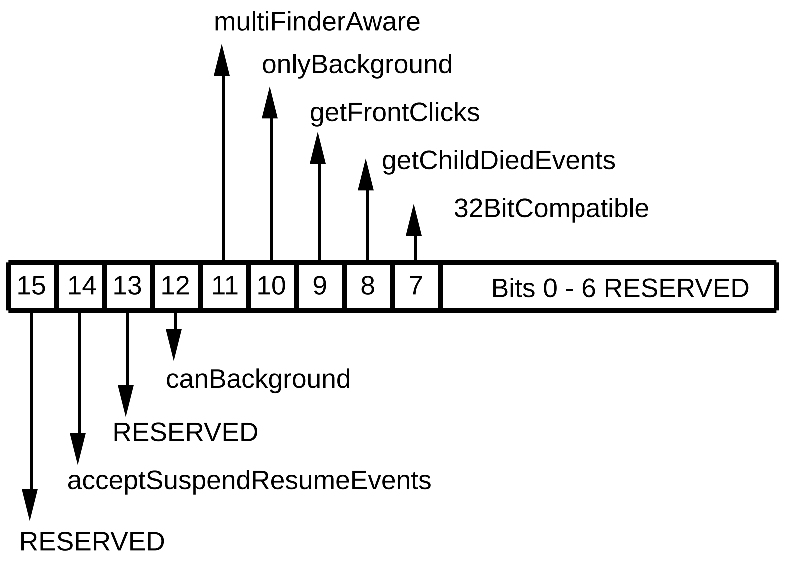

Figure 1–Flags in the 'SIZE' Resource

An application can set the `getFrontClicks` flag if it wants to receive the mouse-up and mouse-down events when the user brings the application's layer to the front. Typically, the user merely wants to bring an application to the front, so it may not be desirable to move the insertion point or start drawing immediately after coming to the foreground. If `getFrontClicks` is set, the mouse click will be passed to the application. If the click was made in the content region of the background application's front window, the application would receive a click in the content region of that window if `getFrontClicks` is set.

Clicking on a window that is behind a window within the same layer will cause the usual event processing (i.e., the mouse-down event will be visible to the application), for which the application will call `_SelectWindow`, bringing the window forward. This is true whether or not the bit is set. Ordinarily these events are not passed to the application, so setting the `getFrontClicks` flag is usually not appropriate. The Finder, however, is one example of an application which has the `getFrontClicks` flag set.

`The getChildDiedEvents` flag is used by SADE to get notification when an application it launched quits or crashes. When you get a `childDiedEvent`, the `message` field of the event record will look like the following:

$FD	Status	Reserved	Reserved

The `Status` parameter will be a system error code if the application crashed, or zero if it quit normally. The `where` field of the event record contains the process identifier (pid) of the quitting process. The `_Launch` trap returns the pid of the newly created application in D0 if the `_Launch` succeeds (if D0 is negative, it contains an OS error code).

Note: Future versions of the Macintosh System Software may operate only in 32-bit mode on 68020 and 68030 machines, so applications which are not 32-bit clean will not function correctly with these machines. The `32BitCompatible` bit will be used in future systems to warn users that running an application which does not have the bit set **may crash their system**. Developers should **not** set this bit unless they have tested their applications on a 32-bit system. Currently, the only 32-bit system available for testing is A/UX, so running under A/UX should be considered the "litmus test" for 32-bit compatibility until newer System Software is available.