F6. Background Processing (Advanced)

One of the biggest differences between FaceIt and competing tools is the degree to which FaceIt, in combination with other modules, automatically handles nearly all window events. Most programmers using FaceIt, for example, will never write a single line of code to handle update events. This aspect of FaceIt programming makes it especially easy for a programmer to implement background processing and switching under MultiFinder or System 7.

What To Do

First, a SIZE resource (like that discussed in the "Finder Resources" topic) with its "Can background" bit set must be made a part of your program file for background processing to work properly. Second, you must add GetNextEvent (or WaitNextEvent) calls to the program code that is to run in the background, and deal properly with the results of these calls. The tricky part of this is in determining how often to call GetNextEvent. If you call GetNextEvent too often, then your background processing will be unnecessarily slowed, whereas too few GetNextEvents makes the top application appear sluggish.

When To Do It

The simplest solutions to the question of whether and how often to call GetNextEvent (or WaitNextEvent) to support background processing and switching are either (a) let the user decide, or (b) make the call after a fixed number of loops or ticks (1/60 seconds) have elapsed. In many cases the background task will involve a loop within which you can place a GetNextEvent call. But rather than slowing things down by calling GetNextEvent every time through the loop, you should instead check a flag, loop index, and/or the current tick count (returned by TickCount) to determine whether it is time to call GetNextEvent. Such a flag can be made a function of whether or not you are running under MultiFinder, and whether or not the user wants to sacrifice processing speed by supporting background processing and switching. The shared variable fEnvFlags will have its second bit set if MultiFinder temporary memory allocation routines are implemented (still a good indicator of whether MultiFinder is in use - see "fRec Record" in ViewIt Guide for more info about fEnvFlags). An example of background processing is presented in the fDemoXY program.

FaceIt's Role

If all you had to do, to support background processing and switching, was decide where and when to call GetNextEvent or WaitNextEvent, then FaceIt wouldn't have much to do with it. The potential programming nightmare, however, comes not in making these calls, but rather in dealing with the Update or SuspendResume (switching) events that your program can be fed while you are deep in your code doing some serious number-crunching. Obviously, you haven't written that code to deal with Update or SuspendResume events! That's where FaceIt comes to the rescue. When you get such an event, you just pass it to FaceIt via DoEvnt to tell it to handle the event. These events usually arise when windows from another application are moved over your program windows, and when the user attempts to switch between applications.

An Example

The following Pascal code fragment illustrates how simple FaceIt makes your handling of Update and SuspendResume events. The example code checks whether GetNextEvent should be called based upon a loop counter, "i", and upon a flag, "doBack", which could be either user-defined and/or made a function of the environment you are running under. If GetNextEvent returns true, and the event is an Update (6) or SuspendResume (15) event, then FaceIt is called to handle the event. The event record gets passed to FaceIt via the fEvent variable in fRec.

```
...
if doBack then
  if (i mod 50 = 0) then
    if GetNextEvent(-1, fEvent) then
      if (fEvent.what=6) or (fEvent.what=15) then
        FaceIt(nil,DoEvnt,0,0,0,0);
...
```

Note that this code works equally well whether the program is the front application or is in the background. In general, your program does not need to be aware of whether it is in the foreground or

background, switched in or switched out. (If you do find a reason for needing to know this, check the fRec variable fSleep. If fSleep = fBackSleep, then you are operating in the background.)

Although the exact logic that surrounds a GetNextEvent call will depend entirely on your programming objectives, the overall approach is always the same: a decision is made whether or not to call GetNextEvent based on some program, user, and/or environmental condition, GetNextEvent gets called, and all Update and SuspendResume events returned are passed to FaceIt via DoEvnt.

Other Events

The flip side of using GetNextEvent (or WaitNextEvent) to support background processing and switching is that this call cannot be used for other purposes without also checking for Update and SuspendResume events. This means that you must either (a) always check for Update & SuspendResume events after calling GetNextEvent, or (b) find a way to avoid using GetNextEvent when you don't want other apps to get background time, nor any switching to occur.

The first approach uses GetNextEvent in a manner like that shown above, but adds one or more statements that check for some other event of interest. A typical use of this approach is to support exiting a time-consuming routine, and a simple event to watch for is an autokey (5) event which occurs when the user holds a key down:

```
if GetNextEvent(-1, fEvent) then
  if (fEvent.what=6) or (fEvent.what=15) then
    FaceIt(nil,DoEvnt,0,0,0,0)
  else if (fEvent.what = 5) then
    [exit time-consuming routine]
```

This approach works well when your primary purpose for calling GetNextEvent is to support background processing.

The second approach, that of avoiding GetNextEvent calls, makes use of toolbox calls such as Button or GetKeys to directly check for mouse clicks or key presses without ever calling GetNextEvent. This approach is better suited for cases where you are interested primarily in the event, and not in supporting background processing. When the desired state is detected, a FlushEvents call should also be made to remove the corresponding event from the event queue so that it does not get processed again by FaceIt. A simple use of Button is shown here, and the example program fDemoXY demonstrates the use of GetKeys.

```
if Button then
  begin
    FlushEvents(62,0); remove spurious events
    [exit time-consuming routine]
  end
```

where the use of "62" when calling FlushEvents removes all mouse and key events that may have accumulated in the event queue.

WaitNextEvent?

GetNextEvent is equivalent to calling WaitNextEvent with a sleep parameter of zero. Zero sleep is, in many cases, a reasonable value to use when the program is in the process of doing some computationally-intensive operation: you're either in the background at the mercy of the System and other applications for processing time, and willing to take all the time you can get, or you're in the foreground in a time-consuming routine that you want to get through as quickly as possible. Thus GetNextEvent works as well as WaitNextEvent for typical uses of background processing.

The time when it makes sense to use a larger sleep value is when your program is in its main event loop, just waiting for an event to occur. But that is FaceIt's job! So FaceIt uses WaitNextEvent in its own event loop, passing it a sleep value equal to fFrontSleep when in the foreground, or fBackSleep when in the background (which have default values of 6 and 8, respectively).