Writing an Idle Function

This section describes how to write an idle function when using the **AESend** or **AEInteractWithUser** function.

When your application sends an Apple event, you can wait for the server application to receive and finish handling the Apple event, or you can continue processing. If your application chooses to continue processing, the **AESend** function returns immediately after using the **Event Manager** to send the event. If your application chooses to wait for the server to handle the event, the **AESend** function does not return until either the server application finishes handling the Apple event or a specified amount of time expires.

Your application specifies its preferences by setting flags in the *sendMode* parameter to **AESend**. Your application can specify kAENoReply if it does not want to receive a reply, kAEQueueReply if it wants to receive the reply in its event queue, or kAEWaitReply if it wants the reply returned in the *reply* parameter of **AESend** and is willing to give up the processor while waiting for the reply.

If your application specifies the kAEWaitReply flag, the **AESend** function calls **WaitNextEvent** on behalf of your application. This yields the processor to other processes, so that the server has an opportunity to receive and process the Apple event sent by your application. While your application is waiting for a reply, it cannot receive events unless it provides an idle function.

If your application provides a pointer to an idle function as a parameter to the **AESend** function, **AESend** calls your idle function whenever an update event, null event, operating-system event, or activate event is received for your application. Your application can process high-level events that it receives while waiting for a reply by providing a reply filter function.

In a similar manner, when your application calls the **AEInteractWithUser** function, your application can also yield the processor. If **AEInteractWithUser** needs to post a notification request to bring your application to the front, your application yields the processor until the user brings your application to the front. To receive events while waiting for the user to bring your application to the front, you must provide an idle function.

If your application provides a pointer to an idle function as a parameter to the **AEInteractWithUser** function, **AEInteractWithUser** calls your idle function whenever an update event, null event, activate event or operating-system event is received for your application.

An idle function must use this syntax:

pascal Boolean **MyIdleFunction(**EventRecord **event*, long **sleepTime*, RgnHandle **mouseRgn*)**

The parameter *theEventRecord* is the event record of the event to process. The *sleepTime* parameter and *mouseRgn* parameter are values that your idle function sets the first time it is called; thereafter they contain the values your function set. Your idle function should return a Boolean value that indicates whether your application wishes to continue waiting. Set the function result to TRUE if your application is no longer willing to wait for a reply from

the server or for the user to bring the application to the front. Set the function result to FALSE if your application is still willing to wait.

The first time your idle function is called, it receives a null event. At this time, you should set the values for the *sleepTime* and *mouseRgn* parameters. These parameters are used in the same way as the *sleep* and *mouseRgn* parameters of the **WaitNextEvent** function. Specify in the *sleepTime* parameter the amount of time (in ticks) during which your application agrees to relinquish the processor if no events are pending for it.

In the *mouseRgn* parameter, you specify a screen region that determines the conditions in which your application is to receive notice of mouse-moved events. Your idle function receives mouse-moved events only if your application is the front application and the mouse strays outside the region you specify.

Your idle function receives only update events, null events, operating-system events, and activate events. When your idle function receives a null event, it can use the idle time to update status reports, animate cursors, or perform similar tasks. If your idle function receives any of the other events, it should handle the event as it normally would if received in its event loop.

The following program shows an example of an idle function that can be used as an idle function for **AESend** or **AEInteractWithUser**. The idle function processes update events, null events, operating-system events, and activate events. The first time the function is called it receives a null event. At this time, it sets the *sleepTime* and *mouseRgn* parameters. The function continues to process events until the server finishes handling the Apple event or the user brings the application to the front.

Your application should implement a method of checking to see if the user wants to cancel. The MyCancelInQueue function in the following program checks the event queue for any instances of Command-period and immediately returns TRUE as its function result if it finds a request to cancel in the event queue.

```
// Assuming inclusion of MacHeaders
#include <AppleEvents.h>

// prototype your routine like this prior to calling it
// NOTE that it uses pascal calling conventions since it is called
// from the AESend routine
pascal Boolean MyIdleFunction(EventRecord *event,
                    long  *sleepTime, RgnHandle *mouseRgn);

// Application global variable to keep track of cursor rgn
RgnHandle gCursorRgn;

pascal Boolean MyIdleFunction(EventRecord *event,
                    long *sleepTime, RgnHandle *mouseRgn)
{
    char    hiByte;
    OSErr  myErr;
```

```
    // Function Prototypes
    Boolean     MyCancelInQueue(void);
    void     DoIdle (void);
    void     AdjustCursor (Point where, RgnHandle cursRgn);
    void     DoEvent (EventRecord *event);


    // the MyCancelInQueue function checks for Command-period
    if ( MyCancelInQueue () )
        return TRUE;

    switch ( event->what ) {
    case updateEvt:
    case activateEvt:   // every idle function should handle
    case app4Evt:       // these kinds of events
        AdjustCursor(event->where, gCursorRgn);
        DoEvent(event);
        break;
    case nullEvent:
        // set the   sleeptime and mouseRgn parameters
        *mouseRgn = gCursorRgn;
        *sleepTime = 10; // use the correct value for your app
        DoIdle();  // the application's idle handling
        break;
    }
    return FALSE;
}
```