Modal Dialog Hints How to get updates to happen properly while in a modal dialog

Modal dialogs boxes have always caused update problems for windows behind dialog windows. Since the **ModalDialog** call makes an internal event call that bypasses the normal event loop in your application, there is always the potential for not knowing that updates have occured for the other windows in your application when you are in a **ModalDialog** loop.

If you have ever written a filter procedure for a modal dialog, you have probably seen this for yourself. Your filter will get a continual stream of update events. These events are not for the dialog, but are for the window behind the dialog, which has not been updated since the modal dialog came up. Since the event has not come through your normal event loop you have probably not serviced the update since you are only concerned about events for your dialog, so it keeps getting resent. The only way for the update to stop is for the update region of the affected window to be cleared, by the **BeginUpdate**/**EndUpdate** calls in your drawing routine.

This problem is exacerbated by screen savers or Balloon Help in System 7.0. If a screen saver becomes active while a modal dialog is up, or if your user has Balloon Help on and part of a behind window is obscured by a balloon, then an update event will be generated for the behind window, and you normally have no way to clear it.

Under System 7 (and System 6 under MultiFinder), if there is an update event pending for your application, no other applications, drivers, control panels, or anything else will get time.

Updates pending for other applications do not cause the problem, they will be handled normally by the application in the background. But updates for the frontmost application must be serviced or the other applications will not get time.

You have two choices in your application to prevent this from happening. The first is to have no other open windows in your application when you open a modal dialog. This is rarely a realistic solution.

The second solution is to provide yourself a mechanism to refresh all your windows from within your modal dialog.

A filter procedure can be used to fix the problem. You will need to add a simple filter procedure to every dialog or alert you bring up in your application. In most cases, it can be the same filter for every dialog, so it is not a great deal of extra code.

The filter procedure needs to have a way to call the drawing procedure for any of your windows. The simplest way would be to include a flag for your drawing procedure in your window record refCon, and have your drawing routine vector based on the value in the refCon, as shown below:

Example

```
#include <Dialogs.h>
```

```
Boolean MyDrawProc(WindowPtr windowToDraw);
void DrawMyClip (WindowPtr windowToDraw);
void DrawMyDoc(WindowPtr windowToDraw);

Boolean MyDrawProc (WindowPtr windowToDraw)
{
Boolean returnVal = true; /* tell Dialog Manager if you handled the update */

switch(GetWRefCon(windowToDraw)){
    case kMyClipboard:
        DrawMyClip(windowToDraw);
        break;
    case kMyDocument:
        DrawMyDoc(windowToDraw);
        break;
    default:
        returnVal = false;
        break;
    }

    return returnVal;
}

/* install the flag when creating the window */
myWindowPtr = GetNewWindow(kMyWindowID, nil,(WindowPtr)-1);
SetWRefCon(myWindowPtr,(long)myDrawingProcFlag);

/* in your filter, the update handling would look something like this... */

if(theEventIn->what == updateEvt &&
                    (WindowPtr) theEventIn->message != myDialogPtr)
    {
    /* if the update is for the dialog box, ignore it since the regular
    ModalDialog function will redraw it as necessary */
    return (MyDrawProc((WindowPtr)theEventIn->message));
    /* go to drawing routine, window will be redrawn if it belongs to app. */
    }
```

The only problem with adding your own filter procedure to a dialog is that the **Dialog Manager** assumes that you are handing more than just updates. Specifically, the **Dialog Manager** assumes that you are handling the standard "return key aliases to item 1" filtering. So, you need to write keystroke handling in the filter yourself.

This is starting to sound complicated, but Apple has provided several new routines in System 7 to address this problem. These new routines allow you to call on the services of the System to track standard keystrokes in your dialog.

You must call the standard filter proc (obtained with a call to **GetStdFilterProc**) for these new calls to work properly. Automatic cursor tracking, default button bordering, and keystroke aliasing for OK and Cancel will only be active if you call the standard filter procedure. Also, these calls are System 7 specific. You cannot use them in previous system versions. Use the **Gestalt Manager** to determine what system you are running, and what

functionality your application can count on being available at run time.

The new System 7 calls are **SetDialogDefaultItem**, **SetDialogCancelItem**, **SetDialogTrackCursor** and **GetStdFilterProc**. Using these calls requires some preparation. After you create your dialog, you need to tell the **Dialog Manager** which items you want as the default and cancel items. The button selected as the cancel item will be toggled by the Escape key or by a Command-period keypress. The button specified as the default will be toggled by the return or enter key, and also will have the standard heavy black border drawn around it. The button will also be hilited when the correct key is hit.

The **SetDialogTrackCursor** call tells the **Dialog Manager** that you have edit lines in your dialog. When you pass a true value, the **Dialog Manager** will constantly check cursor position in your dialog, and change the cursor to an I-Beam when the cursor is over an edit line. So the complete System 7 filter, incorporating update handling and new **Dialog Manager** calls, will look something like this:

Example

```
#include <Dialogs.h>

pascal Boolean myFilter (DialogPtr theDlg, EventRecord *theEvent,
        short *itemHit);


myDialogPtr = GetNewDialog(kMyDialogID, nil, (WindowPtr)-1);
myErr = SetDialogDefaultItem(myDialogPtr,OK);
myErr = SetDialogCancelItem(myDialogPtr,CANCEL);
myErr = SetDialogTrackCursor(myDialogPtr,TRUE);

do {
    ModalDialog(myFilter,&hitItem);
} while(hitItem != OK && hitItem != CANCEL);

/* the filter will look something like this */
pascal Boolean myFilter(DialogPtr currentDialog, EventRecord *theEventIn,
short *theDialogItem)
{
    OSErr       myErr;
    Boolean     returnVal = FALSE;
    WindowPtr   temp;
    ProcPtr    *standardProc;


    if(theEventIn->what==updateEvt &&
            (WindowPtr) theEventIn->message!=currentDialog)
    {
        /* if the update is for the dialog box, ignore it since the regular
            ModalDialog function will redraw it as necessary.
        */
        returnVal = MyDrawProc((WindowPtr) theEventIn->message);
    }
```

```
    else
    {
        /* it wasn't an update, pass it on to the system filter */
        GetPort(&temp);/* save the current port, set to the dialog */
        SetPort(currentDialog);
                        /* necessary to track the edit cursor changes */

        myErr = GetStdFilterProc(&standardProc);
        if(!myErr)
        {
            returnVal = ((ModalFilterProcPtr)standardProc)
                (currentDialog,theEventIn,theDialogItem);

            SetPort(temp);
        }
        return returnVal;
    }
}
```

Since you cannot be sure your users are running System 7, Apple has provided an example filter proc that does roughly the same thing for System 6.0.x users.

Example

```
#include <Dialogs.h>

#define kMyButtonDelay 8

pascal Boolean MyFilter(DialogPtr currentDialog, EventRecord *theEventIn,
short *theDialogItem)
{
    Boolean returnVal = FALSE;
    long    waitTicks;
    short   itemKind;
    Handle  itemHandle;
    Rect    itemRect;

    if(theEventIn->what == updateEvt && theEventIn->message != myDialogPtr)
        {
        /* ignore updates for the dialog box, it will automatically be handled */
        returnVal = MyDrawProc(theEventIn->message);
        }
    else
        {
        /* it wasn't an update, see if it was a keystroke */
        /* check for the return or enter key, and alias it as item 1 */
        /* also include a check here for the escape key aliasing as item 2 */
        /* you may decide not to alias the escape key */
        if((theEventIn->what == keyDown) || (theEventIn->what == autoKey))
            {
            switch(theEventIn->message & charCodeMask){
                case kReturnKey:
```

```
                case kEnterKey:
                    /* change whatever is the current item is to the OK item */
                    *theDialogItem = OK;
            GetDItem(currentDialog,OK,&itemKind,&itemHandle,&itemRect);
                    HiliteControl((ControlHandle)itemHandle,inButton);
                    Delay(kMyButtonDelay,&waitTicks);/* wait about 8 ticks */
                    HiliteControl((ControlHandle)itemHandle,FALSE);
                    returnVal = TRUE;
                    break;
                case kEscKey:
                    *theDialogItem = CANCEL;
        GetDItem(currentDialog,CANCEL,&itemKind,&itemHandle,&itemRect);
                    HiliteControl((ControlHandle)itemHandle,inButton);
                    Delay(kMyButtonDelay,&waitTicks);
                    HiliteControl((ControlHandle)itemHandle,FALSE);
                    returnVal = TRUE;
                    break;
                }
            }
        }
return returnVal;
}
```