About the MovableModal Library

Written by: Marco Piovanelli (piovanel@dsi.unimi.it)November 1993

Introduction

Movable modal dialogs are probably one of the coolest user interface enhancements that came with System 7. Unfortunately, implementing a movable modal dialog according to the guidelines detailed in Inside Macintosh VI may not be trivial. A movable modal dialog box isn't just a modal dialog with a drag bar. In fact, says the Bible:

"Allow your application to run in the background when you display a movable modal dialog box." (IM VI, 2-25)

Given this requirement, your application cannot call `ModalDialog` when a movable modal dialog is in front, because `ModalDialog` disables the application menu and prevents major context switches from occurring. That's too bad, as `ModalDialog` is a very handy call. Wouldn't it be great if a similar call existed for movable modal dialogs too?

The Library

Here is it! The library that comes with this document implements three routines that let you work with movable modal dialogs in a simple way. The main call has this prototype:

```
/* C */
pascal void MovableModalDialog(ModalFilterProcPtr filterProc,
                              short *itemHit);

{ Pascal }
PROCEDURE MovableModalDialog(filterProc: ProcPtr;
                            VAR itemHit: Integer);
```

This routine assumes the frontmost window is your dialog. It handles all events until an enabled dialog item is hit (the item ID is returned in `itemHit`), allows access to the menu bar (even under System 6.0.x) and beeps if the user clicks anywhere outside the dialog or the menu bar. Although you can pass `NULL` in `filterProc`, passing the address of your own filter procedure is strongly recommended. There are two good reasons to do this:

1. If your dialog is dragged around, windows behind it will need to be updated, and this can only be done from within a custom filter proc. In fact, a similar filter proc should be used for **all** modal dialogs (yes, even the non-movable ones and even the Standard File dialogs and the Print Manager dialogs), because things like Balloon Help and Close View can easily invalidate background windows. (see the technical note "M.TB.PendingUpdates").

2. If your application is switched in and out while the dialog is in front, you usually want to know this, so that, for example, you may use the Notification Manager when your application is in background. `MovableModalDialog` passes all OS events to your filter proc.

That's all as for `MovableModalDialog`, but, wait!, that's not all. There are a couple of other routines that the library uses to handle access to the menu bar correctly. The first one is:

```
/* C */
pascal void DisableMenuBar(short editMenuID,
                          short hmnuID);

{ Pascal }
PROCEDURE DisableMenuBar(editMenuID: Integer;
                        hmnuID: Integer);
```

Call this soon **after** you have put up your dialog, but **before** calling `MovableModalDialog`. This routine disables all menus except the "system" menus (the Help and Application menus plus the Keyboard menu and other script-related menus, if any).
If you supply the menu ID of your Edit menu in `editMenuID` and your dialog has edit fields, `DisableMenuBar` enables Cut, Copy and Paste, much like `ModalDialog` does.
Pass the ID of a valid 'hmnu' resource in `hmnuID` if you want menu help string remapping to take place, as described in the technical note "M.TB.MovableModalDialog"; otherwise pass -1 (if the Help Manager isn't available, nothing will happen anyway).

Finally we have:

```
/* C */
pascal void ReEnableMenuBar(void);

{ Pascal }
PROCEDURE ReEnableMenuBar;
```

Call this when you're done with the dialog to re-enable the menus that were disabled by `DisableMenuBar`.

Further Reference:
• *Inside Macintosh*, Volume VI, User Interface Guidelines, Compatibility Guidelines
• M.TB.MovableModalDialog
• M.TB.PendingUpdates