21

FLOATING WINDOWS — MAC OS 8.5 AND EARLIER

Includes Demonstration Program FloatingWindows

Floating Windows

Floating windows are windows which stay in front of all of an application's document windows. They are typically used to display tool, pattern, colour, and other choices to be made available to the user. Examples of floating windows are shown at Fig 1.
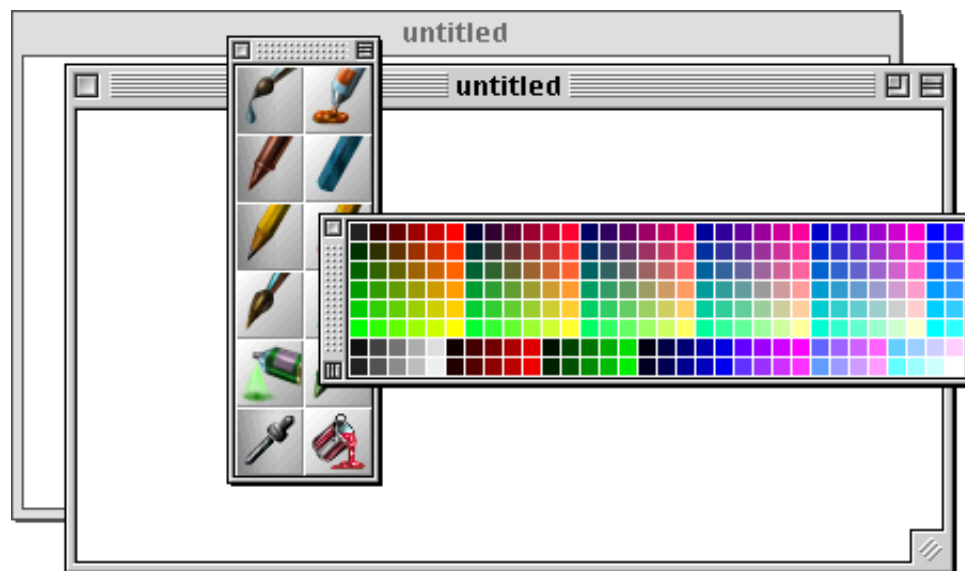

FIG 1 - FLOATING WINDOWS - EXAMPLES

System support for floating windows was introduced with Mac OS 8.5; however, there were bugs in the window activation area which prevented programmers from utilising this inbuilt support. The bugs were eliminated in Mac OS 8.6. Chapter 4B — More on Windows — Mac OS 8.5 Window Manager and the associated demonstration program Windows2 describe and demonstrate system-supported floating windows.

If your application requires floating windows, and is required to run under Mac OS 8.5 or earlier, the application itself will have to provide the necessary support. Application support for floating windows is addressed in this chapter.

Front-To-Back Ordering of On-Screen Objects

The fact that floating windows always remain in front of an application's document windows leads naturally to a consideration of the correct front-to-back ordering of on-screen interface objects. Within an application, this front-to-back ordering should be as follows:

•   Help balloons.

•   Menus.

•   System windows.[1]

•   Modal and movable modal dialog and alert boxes.

•   Floating windows.

•   Document windows and modeless dialog boxes.

Note that floating windows should remain behind modal and movable modal dialog and alert boxes, reflecting the fact that, whatever choices the user can make from a floating window, those choices relate only to operations within the application's document windows and not to operations within modal dialogs and alert boxes.

In terms of front-to-back ordering, floating windows, unlike document windows, are all basically equal. Unless they actually overlap each other, there is no visual cue of any front-to-back ordering as there is with normal windows. Because of this equality, floating windows should almost always appear in the active state. The exception is when a modal or movable modal dialog or alert box is presented to the user. When this occurs, the appearance of all floating windows should be changed to reflect the inactive state. As a further refinement, the content of the window should be dimmed to further suggest to the user that the floating windows are irrelevant to operations within the dialog box or alert box.

Implementing Floating Windows — Considerations

Activate Events

The most significant aspect of implementing floating windows has to do with activate events.

The Single Active Window Problem. The Window Manager was written on the assumption that there is only ever one active window. However, this will not be the case in an application which implements floating windows. (See Fig 1, in which two floating windows and one document window are active at the same time.) Accordingly, you will need to work around how the Window Manager generates activate events and how the Toolbox Event Manager reports them to an application.

The Single Deactivate Event Problem. Because the Window Manager works on the principle that only one window is ever active, only one deactivate event is generated for every activate event. This behaviour will not suffice for an application with floating windows when a modal or movable modal dialog receives the activate event. In that case, a deactivate event is required not only for the frontmost document window but for all of the visible floating windows as well.

These two problems mean that you must not use those Window Manager functions, such as `SelectWindow`, `ShowWindow`, and `HideWindow`, which implicitly generate activate and deactivate events. Instead, you must use lower-level functions like `BringToFront`, `ShowHide`, and `HiliteWindow` to simulate the higher-level calls.

Activate Events and Document Windows. Other cases that the Window Manager does not handle well occur when the frontmost document window is closed or when a new document window is created in

---

[1] System windows are windows which can appear in an application's window list but which are not directly created by the application. These windows appear in front of all windows created by the application. An example of a system window is a notification alert box.

front of other document windows. If floating windows are present, these document windows do not receive the needed activate and deactivate events, since the application is essentially removing or creating windows in the middle of the window list. Accordingly, your application must itself manage the activation and deactivation of the relevant windows.

Activate Events and Modal Windows. When a modal window is to appear, your application will need to deactivate all visible floating windows and the active document window. When the user dismisses the modal window, your application must re-activate each of those windows.

These considerations require that you subvert the system software's normal window activation/deactivation activities and divide the window list into two sections, specifically, the section occupied by the floating windows (which must always be at the beginning of the overall list) and the section occupied by the document windows.

Opening, Closing, Showing and Hiding

Floating windows should be opened at application launch and should remain open until the application is closed. Since **Open…** and **Close** items in the **File** menu should apply only to document windows, items in some other appropriate menu should be provided to allow the user to toggle each floating window between the hidden and showing state.

A floating window's close box should simply hide the window, not close it. For that reason, the close box in floating windows should be conceived of as a "hide" box rather than as a go-away box.

Application in the Background. Floating windows should be hidden by the owner application when that application receives a suspend event. This is to avoid user confusion arising from one application's floating windows being visible when another application is in the foreground. The floating windows should be shown again only when the application receives a subsequent resume event.

Implementing Floating Windows — Substitute and Supporting Functions

Implementing floating windows in an application basically involves providing a number of application-defined substitute and supporting functions, many of which perform the necessary subversion of the system software's normal window activation/deactivation activities and treat the window list as comprising separate document windows and floating windows sections.

The following reflects the implementation methodology used in the demonstration program associated with this chapter.

Main Substitute Functions

The substitute functions are those functions used in lieu of the Macintosh Toolbox functions that would ordinarily be used in a non-floating windows environment. The main substitute functions[2], the actions performed by those functions, and the Toolbox calls they replace, are as follows:

Substitute	Replaces	Actions Performed
FW_GetNewCWindow	GetNewCWindow	Create floating and document windows based on a resource template.
		When a new floating window is opened, bring that window to the very front of any existing windows.
		When a new document window is opened, move that window to immediately behind the last floating window, and in front of any document windows which may already be open.
FW_DisposeWindow	DisposeWindow	Dispose of document windows.
		Activate the next document window in the window list, and call DisposeWindow to remove the specified document window from the screen and the window list and discard all it data storage.

---

[2] The names of the replacement and supplementary functions shown in the following are, of course, purely arbitrary. You may use whatever names you like. The names shown are those used in the demonstration program.

FW_SelectWindow	SelectWindow	Bring the window (floating or document) in which the mouse-down occurred as far forward in the window list as it should come. If it is a floating window, makes it the absolute frontmost window. If it is a document window, make it the frontmost window behind the floating windows.
FW_HideWindow	HideWindow	Hide the specified window. As with HideWindow, if the frontmost (floating or document) window is to be hidden, place it behind the window immediately behind it in its section of the window list so that, when it is shown again, it will no longer be frontmost window in its section.
FW_ShowWindow	ShowWindow	Show the specified window. As with ShowWindow, show the window without changing its position in the window list. If the window being shown is the frontmost document window, deactivate the window behind it, and activate the window being shown. If the specified window is a floating window, and if a modal or movable modal dialog box is present, show the window in the inactive state.
FW_DragWindow	DragWindow	Drag the specified window around, ensuring that, if it is a document window, it remains behind the floating windows. As with DragWindow, do not bring the window forward if the Command key is held down during the drag.

Additional Substitute Functions

The demonstration program associated with this chapter uses the refCon field of all document and floating window window structures to store a handle to a special structure containing information relevant to window management in the floating windows environment. This means that the use of the window structure's refCon field is denied to the application for other purposes (such as, for example, storing a handle to a document structure).

To compensate for this, a reference constant field is included in the special structure containing information relevant to window management. This field may be used to store the value that you would ordinarily assign to the window structure's refCon field. The following additional substitute functions pertain to assigning a value to, and retrieving that value from, this field:

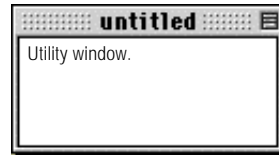| *Substitute*	*Replaces*	*Actions Performed*
FW_SetWRefCon	SetWRefCon	Assign a value to the reference constant field of a structure whose handle is assigned to the refCon field of a window's window structure.
FW_GetWRefCon	GetWRefCon	Get the value in the reference constant field of a structure whose handle is assigned to the refCon field of a window's window structure.

Supporting Functions

The main supporting functions, and the actions performed by those functions, are as follows:

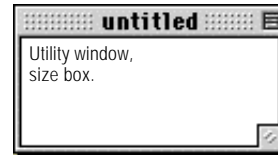| *Function*	*Actions Performed*
FW_doSuspendEvent	Hide any floating windows, and unhighlight and deactivate the frontmost document window. (Call this function when the application receives a suspend event.)
FW_doResumeEvent	Show all floating windows which were visible when the application was sent to the background, and highlight and activate the front document window. (Call this function when the application receives a suspend event.)
FW_deactivateFloatsAndFirstDocWin	Unhighlight and deactivate any visible floating windows and the frontmost document window. (Call this function immediately before a modal or movable modal dialog or alert box is invoked.)
FW_activateFloatsAndFirstDocWin	Highlight and activate those windows which were visible and activated before FW_deactivateFloatersAndFirstDocWin was called. (Call this function immediately after an alert or modal dialog box is dismissed.)

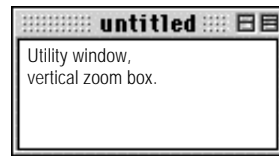|	
`FW_findFrontNonFloatWindow`	Find the first visible non-floating window in the window list.
	(Call this function when you need the pointer to the frontmost document window or when you need to determine whether any document windows are currently open.)
`FW_validateWindowList`	Ensure that all floating windows are in front of all document windows.
	(Call this function, as a safety measure, immediately prior to opening and closing document windows.)

Floating Window Types

Figs 2 and 3 show the sixteen available window types for floating windows and the constants that represent those types.
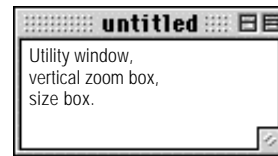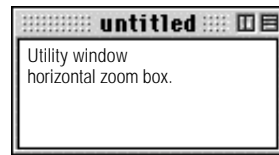

FIG 2 - WINDOW TYPES FOR FLOATING WINDOWS (TITLE BAR AT TOP)

kWindowFloatSideProc

kWindowFloatSideGrowProc

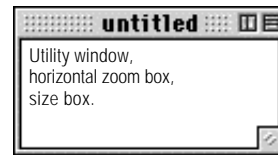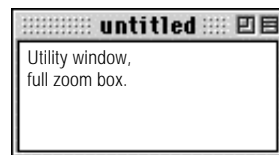
kWindowFloatSideVertZoomProc

kWindowFloatSideVertZoomGrowProc

kWindowFloatSideHorizZoomProc

kWindowFloatSideHorizZoomGrowProc

kWindowFloatSideFullZoomProc

kWindowFloatSideFullZoomGrowProc

FIG 3 - WINDOW TYPES FOR FLOATING WINDOWS (TITLE BAR AT SIDE)

Window Definition IDs

The WDEF resource IDs for the floating window types are 66, and 67. The window definition IDs are as follows:

WDEF Resource ID	Variation Code	Window Definition ID (Value)	Window Definition ID (Constant)
66	1	66 * 16 + 1 = 1057	kWindowFloatProc
66	3	66 * 16 + 3 = 1059	kWindowFloatGrowProc
66	5	66 * 16 + 5 = 1061	kWindowFloatVertZoomProc
66	7	66 * 16 + 7 = 1063	kWindowFloatVertZoomGrowProc
66	9	66 * 16 + 9 = 1065	kWindowFloatHorizZoomProc
66	11	66 * 16 + 11 = 1067	kWindowFloatHorizZoomGrowProc
66	13	66 * 16 + 13 = 1069	kWindowFloatFullZoomProc
66	15	66 * 16 + 15 = 1071	kWindowFloatFullZoomGrowProc
67	1	67 * 16 + 1 = 1073	kWindowFloatSideProc
67	3	67 * 16 + 3 = 1075	kWindowFloatSideGrowProc
67	5	67 * 16 + 5 = 1077	kWindowFloatSideVertZoomProc
67	7	67 * 16 + 7 = 1079	kWindowFloatSideVertZoomGrowProc
67	9	67 * 16 + 9 = 1081	kWindowFloatSideHorizZoomProc
67	11	67 * 16 + 11 = 1083	kWindowFloatSideHorizZoomGrowProc
67	13	67 * 16 + 13 = 1085	kWindowFloatSideFullZoomProc
67	15	67 * 16 + 15 = 1087	kWindowFloatSideFullZoomGrowProc
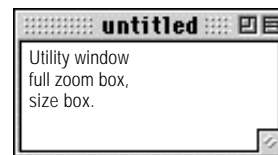
Floating Windows Library Functions

In the accompanying demonstration program files, the source code pertaining to implementing a floating windows environment has been compiled as a library. The following are the prototypes for those functions in the library that need to be called from an application.

Main Substitute Functions

```
OSErr      FW_GetNewCWindow(WindowPtr *windowPtr,SInt16 windResourceID,WindowPtr behind,
           ActivateHandlerUPP activeFunctionUPP,Boolean isFloater)
void       FW_DisposeWindow(WindowPtr windowPtr)
void       FW_SelectWindow(WindowPtr windowPtr)
void       FW_HideWindow(WindowPtr windowPtr)
void       FW_ShowWindow(WindowPtr windowPtr)
void       FW_DragWindow(WindowPtr windowPtr,Point startPoint,const Rect *draggingBounds)
```

Additional Substitute Functions

```
SInt32     FW_GetWRefCon(WindowPtr windowPtr)
void       FW_SetWRefCon(WindowPtr windowPtr,SInt32 refCon)
```

Supporting Functions

```
void       FW_doSuspendEvent(void)
void       FW_doResumeEvent(void)
void       FW_activateFloatsAndFirstDocWin(void)
void       FW_deactivateFloatsAndFirstDocWin(void)
WindowPtr  FW_findFrontNonFloatWindow(void)
void       FW_validateWindowList(void)
```

Demonstration Program

```
// #####################################################################################
// FloatingWindows.c
// #####################################################################################
//
// This program utilises custom libraries titled FloatingLib68K (for the 68K project) and
// FloatingLibPPC (for the PowerPC project), for which the source code is provided.
// At startup, it open two floating windows and a document window.  Subsequently,
// document windows, a modal alert box, a movable modal dialog, and the Standard File
// open dialog box may be opened and closed, and the floating windows may be hidden and
// shown, so as to prove correct behaviour of all windows in a floating windows
// environment.
//
// The program utilises the following resources:
//
// •   An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit and Floaters menus
//     (preload, non-purgeable).
//
// •   'WIND' resources (purgeable) (initially not visible) for the document and floating
//     windows.
//
// •   An 'ALRT' resource (purgeable), and associated 'DITL', 'alrx', and 'dftb' resources
//     (purgeable), for an alert invoked by the user choosing the About FloatingWindows
//     item in the Apple menu.
//
// •   A 'DLOG' resource (purgeable), and associated 'DITL', 'dlgx', and 'dftb' resources
//     (purgeable), for a movable modal dialog box invoked when the user chooses the Find
//     item in the Edit menu.
//
// •   'PICT' resources (non-purgeable) containing pictures to be drawn in the floating
//     windows.
//
// •   A 'STR# ' resource (purgeable) containing error strings for an error alert box.
//
// •   An icon family (purgeable), which provides an icon drawn in the document windows.
//
// •   A 'SIZE' resource with the acceptSuspendResumeEvents, doesActivateOnFGSwitch, and
//     is32BitCompatible flags set.
//
```

```
// ############################################################################

// ……………………………………………………………………………………………………………………………………………………………………………………………………………… includes

#include <Appearance.h>
#include <Devices.h>
#include <StandardFile.h>
#include <ToolUtils.h>

// ……………………………………………………………………………………………………………………………………………………………………………………………………………… defines

#define rMenuBar                128
#define mApple                  128
#define   iAbout                1
#define mFile                   129
#define   iNew                  1
#define   iOpen                 2
#define   iClose                4
#define   iQuit                 12
#define mEdit                   130
#define   iFind                 9
#define mFloaters               131
#define   iColours              1
#define   iTools                2
#define rDocumentWindow         128
#define rFloaterColoursWindow   129
#define rFloaterToolsWindow     130
#define rColoursPictActive      128
#define rColoursPictInactive    129
#define rToolsPictActive        130
#define rToolsPictInactive      131
#define rAboutAlert             128
#define rFindDialog             129
#define rIconSuite              128
#define rErrorStrings           128

#define kFloaterKind                    7
#define eWindowNotCreatedError          -2
#define eInvalidWindowOrderingError     -3

// ……………………………………………………………………………………………………………………………………………………………………………………………………………… typedefs

typedef pascal void (*ActivateHandlerProcPtr)
                    (WindowPtr theWindow,Boolean activateWindow);
typedef struct
{
  Handle iconSuiteHdl;
} docStruc;

typedef docStruc **docStrucHandle;

// ……………………………………………………………………………………………………………………………………………………………………………………………………………… global variables

Boolean   gDone;
Boolean   gInBackground;
WindowPtr gColoursFloaterPtr;
WindowPtr gToolsFloaterPtr;
Handle    gIconSuiteHdl;

// ……………………………………………………………………………………………………………………………………………………………………………………………………………… function prototypes

void        main                        (void);
void        doInitManagers              (void);
void        doOpenDocumentWindow        (void);
void        doOpenFloatingWindows       (void);
void        doEvents                    (EventRecord *);
void        doMouseDown                 (EventRecord *);
void        doUpdate                    (EventRecord *eventStrucPtr);
pascal void doActivateWindow            (WindowPtr, Boolean);
pascal void doActivateColoursFloater    (WindowPtr, Boolean);
pascal void doActivateToolsFloater      (WindowPtr, Boolean);
void        doAdjustMenus               (void);
void        doMenuChoice                (SInt32);
void        doFloatersMenu              (SInt16 theItem);
void        doAboutBox                  (void);
void        doStandardFileOpen          (void);
```

```
void          doOpenFindDialog                    (void);
void          doCloseDocWindow                     (void);
void          doDrawWindowContent                  (WindowPtr,Boolean);
void          doErrorAlert                         (OSErr);

// ………………………………………………………… function prototypes - functions in floating windows library

OSErr         FW_GetNewCWindow                     (WindowPtr *,SInt16,WindowPtr,
                                                    ActivateHandlerProcPtr,Boolean);
void          FW_DisposeWindow                     (WindowPtr);
void          FW_SelectWindow                      (WindowPtr);
void          FW_HideWindow                        (WindowPtr);
void          FW_ShowWindow                        (WindowPtr);
void          FW_DragWindow                        (WindowPtr, Point, const Rect *);
void          FW_doSuspendEvent                    (void);
void          FW_doResumeEvent                     (void);
void          FW_activateFloatsAndFirstDocWin      (void);
void          FW_deactivateFloatsAndFirstDocWin    (void);
WindowPtr     FW_findFrontNonFloatWindow           (void);
void          FW_validateWindowList                (void);
SInt32        FW_GetWRefCon                        (WindowPtr);
void          FW_SetWRefCon                        (WindowPtr, SInt32);

// ############################################################################## main

void  main()
{
  Handle       menubarHdl;
  MenuHandle   menuHdl;
  EventRecord  eventStructure;

  // …………………………………………………………………………………………………………………………………………………………………… initialise managers

  doInitManagers();

  // …………………………………………………………………………………………………………………………………………… set up menu bar and menus

  menubarHdl = GetNewMBar(rMenuBar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  menuHdl = GetMenuHandle(mApple);
  if(menuHdl == NULL)
    ExitToShell();
  else
    AppendResMenu(menuHdl,'DRVR');

  // ………………………………………………………………………………………………………………………………………………………………… get icon suite

  GetIconSuite(&gIconSuiteHdl,rIconSuite,kSelectorAllLargeData);

  // ………………………………………………………………………………………………………………………………………………………………… set up windows

  doOpenDocumentWindow();
  doOpenFloatingWindows();

  // …………………………………………………………………………………………………………………………………………………………… enter event loop

  gDone = false;

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&eventStructure,1500,NULL))
      doEvents(&eventStructure);
  }
}

// ############################################################### doInitManagers

void  doInitManagers(void)
{
  MaxApplZone();
  MoreMasters();
```

```
   InitGraf(&qd.thePort);
   InitFonts();
   InitWindows();
   InitMenus();
   TEInit();
   InitDialogs(NULL);

   InitCursor();
   FlushEvents(everyEvent,0);

   RegisterAppearanceClient();
}

// ########################################################### doOpenDocumentWindow

void  doOpenDocumentWindow(void)
{
   WindowPtr       windowPtr;
   OSErr           osError;
   docStrucHandle  docStrucHdl;

   FW_validateWindowList();

   osError = FW_GetNewCWindow(&windowPtr,rDocumentWindow,(WindowPtr) -1,
                              (ActivateHandlerProcPtr) &doActivateWindow,false);
   if(osError == noErr)
   {
      if(!(docStrucHdl = (docStrucHandle) NewHandle(sizeof(docStruc))))
         ExitToShell();
      FW_SetWRefCon(windowPtr,(SInt32) docStrucHdl);
      (*docStrucHdl)->iconSuiteHdl = gIconSuiteHdl;

      FW_ShowWindow(windowPtr);
   }
   else
      doErrorAlert(osError);
}

// ########################################################### doOpenFloatingWindows

void  doOpenFloatingWindows(void)
{
   OSErr     osError;
   PicHandle thePicture;

   osError = FW_GetNewCWindow(&gColoursFloaterPtr,rFloaterColoursWindow,(WindowPtr) -1,
                              (ActivateHandlerProcPtr) &doActivateColoursFloater,true);
   if(osError == noErr)
   {
      thePicture = GetPicture(rColoursPictActive);
      SetWindowPic(gColoursFloaterPtr,thePicture);
      FW_ShowWindow(gColoursFloaterPtr);
   }
   else
      doErrorAlert(osError);

   osError = FW_GetNewCWindow(&gToolsFloaterPtr,rFloaterToolsWindow,(WindowPtr) -1,
                              (ActivateHandlerProcPtr) &doActivateToolsFloater,true);
   if(osError == noErr)
   {
      thePicture = GetPicture(rToolsPictActive);
      SetWindowPic(gToolsFloaterPtr,thePicture);
      FW_ShowWindow(gToolsFloaterPtr);
   }
   else
      doErrorAlert(osError);
}

// ################################################################## doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
   SInt8 charCode;

   switch (eventStrucPtr->what)
   {
```

```
    case mouseDown:
      doMouseDown(eventStrucPtr);
      break;

    case keyDown:
    case autoKey:
      charCode = eventStrucPtr->message & charCodeMask;
      if((eventStrucPtr->modifiers & cmdKey) != 0)
      {
        doAdjustMenus();
        doMenuChoice(MenuEvent(eventStrucPtr));
      }
      break;

    case updateEvt:
      doUpdate(eventStrucPtr);
      break;

    case osEvt:
      switch((eventStrucPtr->message >> 24) & 0x000000FF)
      {
        case suspendResumeMessage:
          if(eventStrucPtr->message & resumeFlag)
          {
            gInBackground = false;
            FW_doResumeEvent();
          }
          else
          {
            gInBackground = true;
            FW_doSuspendEvent();
          }
          break;
      }
      break;
  }
}

// ######################################################################## doMouseDown

void  doMouseDown(EventRecord *eventStrucPtr)
{
  SInt16      partCode;
  WindowPtr   windowPtr;
  WindowPtr   frontWindowPtr;
  Rect        growRect;
  UInt32      newSize;

  partCode = FindWindow(eventStrucPtr->where,&windowPtr);

  switch (partCode)
  {
    case inMenuBar:
      doAdjustMenus();
      doMenuChoice(MenuSelect(eventStrucPtr->where));
      break;

    case inContent:
      frontWindowPtr = FW_findFrontNonFloatWindow();
      if(windowPtr != frontWindowPtr)
        FW_SelectWindow(windowPtr);
      break;

    case inDrag:
      FW_DragWindow(windowPtr,eventStrucPtr->where,&qd.screenBits.bounds);
      break;

    case inGrow:
      growRect = qd.screenBits.bounds;
      growRect.top  = 80;
      growRect.left = 160;
      newSize = GrowWindow(windowPtr,eventStrucPtr->where,&growRect);
      if(newSize != 0)
        SizeWindow(windowPtr,LoWord(newSize),HiWord(newSize),true);
      break;
```

```
      case inGoAway:
        if(TrackGoAway(windowPtr,eventStrucPtr->where))
        {
          if(((WindowPeek) windowPtr)->windowKind == kFloaterKind)
            FW_HideWindow(windowPtr);
          else
            FW_DisposeWindow(windowPtr);
        }
        break;

      case inZoomIn:
      case inZoomOut:
        if(TrackBox(windowPtr,eventStrucPtr->where,partCode))
        {
          SetPort(windowPtr);
          EraseRect(&windowPtr->portRect);
          ZoomWindow(windowPtr,partCode,false);
        }
        break;
  }
}

// ######################################################################## doUpdate

void  doUpdate(EventRecord *eventStrucPtr)
{
  WindowPtr windowPtr;

  windowPtr = (WindowPtr) eventStrucPtr->message;
  SetPort(windowPtr);

  BeginUpdate(windowPtr);

  EraseRgn(windowPtr->visRgn);
  doDrawWindowContent(windowPtr,((WindowPeek) windowPtr)->hilited);

  EndUpdate(windowPtr);
}

// ################################################################### doActivateWindow

pascal void  doActivateWindow(WindowPtr windowPtr,Boolean becomingActive)
{
  GrafPtr oldPort;

  GetPort(&oldPort);
  SetPort(windowPtr);

  doDrawWindowContent(windowPtr,becomingActive);

  SetPort(oldPort);
}

// ############################################################## doActivateColoursFloater

pascal void  doActivateColoursFloater(WindowPtr theWindow,Boolean becomingActive)
{
  GrafPtr   oldPort;
  PicHandle thePicture;
  SInt16    pictResourceID;

  GetPort(&oldPort);
  SetPort(theWindow);

  if(becomingActive)
    pictResourceID = rColoursPictActive;
  else
    pictResourceID = rColoursPictInactive;
  thePicture = GetPicture(pictResourceID);

  DrawPicture(thePicture,&((*thePicture)->picFrame));
  SetWindowPic(theWindow,thePicture);

  if(becomingActive == false)
    SetPort(oldPort);
}
```

```
// ############################################################## doActivateToolsFloater

pascal void  doActivateToolsFloater(WindowPtr theWindow,Boolean becomingActive)
{
  GrafPtr   oldPort;
  PicHandle thePicture;
  SInt16    pictResourceID;

  GetPort(&oldPort);
  SetPort(theWindow);

  if(becomingActive)
    pictResourceID = rToolsPictActive;
  else
    pictResourceID = rToolsPictInactive;
  thePicture = GetPicture(pictResourceID);

  DrawPicture(thePicture,&((*thePicture)->picFrame));
  SetWindowPic(theWindow,thePicture);

  if(becomingActive == false)
    SetPort(oldPort);
}

// ##################################################################### doAdjustMenus

void  doAdjustMenus(void)
{
  MenuHandle  fileMenuHdl, floatMenuHdl;
  Boolean     isVisible;

  fileMenuHdl = GetMenuHandle(mFile);
  floatMenuHdl = GetMenuHandle(mFloaters);

  if(FW_findFrontNonFloatWindow() == NULL)
    DisableItem(fileMenuHdl,iClose);
  else
    EnableItem(fileMenuHdl,iClose);

  isVisible = ((WindowPeek) gColoursFloaterPtr)->visible;
  if(isVisible)
    CheckItem(floatMenuHdl,iColours,true);
  else
    CheckItem(floatMenuHdl,iColours,false);

  isVisible = ((WindowPeek) gToolsFloaterPtr)->visible;
  if(isVisible)
    CheckItem(floatMenuHdl,iTools,true);
  else
    CheckItem(floatMenuHdl,iTools,false);

  DrawMenuBar();
}

// ###################################################################### doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  SInt16    menuID, menuItem;
  Str255    itemName;
  SInt16    daDriverRefNum;

  menuID = HiWord(menuChoice);
  menuItem = LoWord(menuChoice);

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mApple:
      if(menuItem == iAbout)
        doAboutBox();
      else
      {
```

```
        GetMenuItemText(GetMenuHandle(mApple),menuItem,itemName);
        daDriverRefNum = OpenDeskAcc(itemName);
      }
      break;

    case mFile:
      switch(menuItem)
      {
        case iNew:
          doOpenDocumentWindow();
          break;

        case iOpen:
          doStandardFileOpen();
          break;

        case iClose:
          doCloseDocWindow();
          break;

        case iQuit:
          gDone = true;
          break;
      }
      break;

    case mEdit:
      if(menuItem == iFind)
        doOpenFindDialog();
      break;

    case mFloaters:
      doFloatersMenu(menuItem);
      break;
  }

  HiliteMenu(0);
}

// ################################################################## doFloatersMenu

void  doFloatersMenu(SInt16 menuItem)
{
  WindowPtr floaterWindowPtr;
  Boolean   isVisible;

  if(menuItem == iColours)
    floaterWindowPtr = gColoursFloaterPtr;
  else if(menuItem == iTools)
    floaterWindowPtr = gToolsFloaterPtr;

  isVisible = ((WindowPeek) floaterWindowPtr)->visible;

  if(isVisible)
    FW_HideWindow(floaterWindowPtr);
  else
    FW_ShowWindow(floaterWindowPtr);
}

// ##################################################################### doAboutBox

void  doAboutBox()
{
  SInt16  alertResult;

  FW_deactivateFloatsAndFirstDocWin();

  alertResult = Alert(rAboutAlert,NULL);

  FW_activateFloatsAndFirstDocWin();
}

// ################################################################ doStandardFileOpen

void  doStandardFileOpen()
{
```

```
  SFTypeList        fileTypes;
  StandardFileReply fileReply;

  fileTypes[0] = '****';

  FW_deactivateFloatsAndFirstDocWin();

  StandardGetFile(NULL,-1,fileTypes,&fileReply);

  FW_activateFloatsAndFirstDocWin();
}

// ################################################################## doOpenFindDialog

void  doOpenFindDialog(void)
{
  DialogPtr dialogPtr;
  SInt16    itemHit;

  FW_deactivateFloatsAndFirstDocWin();

  dialogPtr = GetNewDialog(rFindDialog,NULL,(WindowPtr) -1);
  SetDialogDefaultItem(dialogPtr,kStdOkItemIndex);

  do
  {
    ModalDialog(NULL,&itemHit);
  } while((itemHit != kStdOkItemIndex) && (itemHit != kStdCancelItemIndex));

  DisposeDialog(dialogPtr);

  FW_activateFloatsAndFirstDocWin();
}

// ################################################################## doCloseDocWindow

void  doCloseDocWindow()
{
  WindowPtr      windowPtr;
  docStrucHandle  docStrucHdl;

  FW_validateWindowList();

  windowPtr = FW_findFrontNonFloatWindow();

  if(windowPtr != NULL)
  {
    docStrucHdl = (docStrucHandle) (FW_GetWRefCon(windowPtr));;
    DisposeHandle((Handle) docStrucHdl);

    FW_DisposeWindow(windowPtr);
  }
}

// ################################################################ doDrawWindowContent

void  doDrawWindowContent(WindowPtr windowPtr,Boolean isActive)
{
  docStrucHandle  docStrucHdl;
  Rect            theRect;
  Handle          iconSuiteHdl;

  docStrucHdl = (docStrucHandle) (FW_GetWRefCon(windowPtr));
  iconSuiteHdl = (*docStrucHdl)->iconSuiteHdl;

  SetRect(&theRect,5,5,37,37);

  if(isActive)
    PlotIconSuite(&theRect,kAlignNone,kTransformNone,iconSuiteHdl);
  else
    PlotIconSuite(&theRect,kAlignNone,kTransformDisabled,iconSuiteHdl);
}

// ###################################################################### doErrorAlert

void  doErrorAlert(OSErr errorCode)
```

```
{
   AlertStdAlertParamRec paramRec;
   Str255                errorString;
   SInt16                itemHit;

   paramRec.movable       = true;
   paramRec.helpButton    = false;
   paramRec.filterProc    = NULL;
   paramRec.defaultText   = (StringPtr) kAlertDefaultOKText;
   paramRec.cancelText    = NULL;
   paramRec.otherText     = NULL;
   paramRec.defaultButton = kAlertStdAlertOKButton;
   paramRec.cancelButton  = 0;
   paramRec.position      = kWindowDefaultPosition;

   if(errorCode == eWindowNotCreatedError)
     GetIndString(errorString,rErrorStrings,1);
   else if(errorCode == eInvalidWindowOrderingError)
     GetIndString(errorString,rErrorStrings,2);

   StandardAlert(kAlertStopAlert,errorString,NULL,&paramRec,&itemHit);
   ExitToShell();
}

// #############################################################################
// FloatingLib.c
// #############################################################################

// ……………………………………………………………………………………………………………………………………………………………………………………………………………… includes

#include <LowMem.h>

// ……………………………………………………………………………………………………………………………………………………………………………………………………………… defines

#define kFloaterKind                 7
#define eWindowNotCreatedError      -2
#define eInvalidWindowOrderingError -3

#define CallActivateHandler(userRoutine,theWindow,activateWindow) \
                       (*userRoutine) (theWindow,activateWindow)

// ……………………………………………………………………………………………………………………………………………………………………………………………………………… typedefs

typedef pascal void (*ActivateHandlerProcPtr)
                    (WindowPtr theWindow,Boolean activateWindow);

typedef struct
{
   ActivateHandlerProcPtr  activateHandler;
   Boolean                 wasVisible;
   SInt32                  replacementRefCon;
} refConExtend;

typedef refConExtend **refConExtendHandle;

// ……………………………………………………………………………………………………………………………………… function prototypes

OSErr           FW_GetNewCWindow                 (WindowPtr *,SInt16,WindowPtr,
                                                   ActivateHandlerProcPtr,Boolean);
void            FW_DisposeWindow                 (WindowPtr);
void            FW_SelectWindow                  (WindowPtr);
void            FW_HideWindow                    (WindowPtr);
void            FW_ShowWindow                    (WindowPtr);
void            FW_DragWindow                    (WindowPtr,Point,const Rect *);
void            FW_doSuspendEvent                (void);
void            FW_doResumeEvent                 (void);
void            FW_deactivateFloatsAndFirstDocWin (void);
void            FW_activateFloatsAndFirstDocWin  (void);
WindowPtr       FW_findFrontNonFloatWindow       (void);
void            FW_validateWindowList            (void);
SInt32          FW_GetWRefCon                    (WindowPtr);
void            FW_SetWRefCon                    (WindowPtr,SInt32);
void            activateWindow                   (WindowPtr);
void            deactivateWindow                 (WindowPtr);
void            highlightAndActivateWindow       (WindowPtr,Boolean);
WindowPtr       findLastFloatingWindow           (void);
```

```
OSErr                    checkWindowOrdering          (Boolean,WindowPtr *);
WindowPtr                getNextVisibleWindow         (WindowPtr);
Boolean                  getWasVisible                (WindowPtr);
void                     setWasVisible                (WindowPtr,Boolean);
Boolean                  isWindowModal                (WindowPtr);
void                     bringFloatersToFront         (WindowPtr);
Boolean                  isFrontProcess               (void);
Boolean                  getIsWindowVisible           (WindowPtr);
WindowPtr                getNextWindow                (WindowPtr);
SInt16                   getWindowKind                (WindowPtr);
RgnHandle                getStructureRegion           (WindowPtr);
RgnHandle                getContentRegion             (WindowPtr);
ActivateHandlerProcPtr   getActivateHandler           (WindowPtr);
void                     setActivateHandler           (WindowPtr,ActivateHandlerProcPtr);

// ################################################################ FW_GetNewCWindow

OSErr  FW_GetNewCWindow(WindowPtr *windowPtr,SInt16 windResourceID,WindowPtr behind,
                        ActivateHandlerProcPtr activeHandler,Boolean isFloater)
{
  OSErr               result;
  refConExtendHandle  refConExtendHdl;
  WindowPtr           newWindowPtr;
  SInt32              replacementRefCon;
  Boolean             isVisible;

  *windowPtr = NULL;

  result = checkWindowOrdering(isFloater,&behind);
  if(result != noErr)
    return result;

  refConExtendHdl = (refConExtendHandle) NewHandle(sizeof(refConExtend));

  if((result = MemError()) == noErr)
  {
    newWindowPtr = GetNewCWindow(windResourceID,NULL,behind);

    if(newWindowPtr != NULL)
    {
      *windowPtr = newWindowPtr;

      replacementRefCon = GetWRefCon(newWindowPtr);
      SetWRefCon(newWindowPtr,(SInt32) refConExtendHdl);
      FW_SetWRefCon(newWindowPtr,replacementRefCon);

      setActivateHandler(newWindowPtr,activeHandler);

      if(isFloater)
      {
        ((WindowPeek) newWindowPtr)->windowKind = kFloaterKind;

        isVisible = ((WindowPeek) newWindowPtr)->visible;

        if(isVisible)
        {
          if(!isFrontProcess())
          {
            ShowHide(newWindowPtr,false);
            setWasVisible(newWindowPtr,true);
          }
        }
      }
    }
    else
    {
      DisposeHandle((Handle) refConExtendHdl);
      result = eWindowNotCreatedError;
    }
  }
  return result;
}

// ################################################################ FW_DisposeWindow

void  FW_DisposeWindow(WindowPtr windowPtr)
```

```
{
  refConExtendHandle  refConExtendHdl;

  if(getIsWindowVisible(windowPtr))
    FW_HideWindow(windowPtr);

  refConExtendHdl = (refConExtendHandle) GetWRefCon(windowPtr);
  if(refConExtendHdl != NULL)
    DisposeHandle((Handle) refConExtendHdl);

  DisposeWindow(windowPtr);
}

// ################################################################## FW_SelectWindow

void  FW_SelectWindow(WindowPtr windowPtr)
{
  Boolean    frontProcess, isFloatingWindow;
  WindowPtr currentFrontWindowPtr, lastFloatingWindowPtr;

  frontProcess = isFrontProcess();

  if(getWindowKind(windowPtr) == kFloaterKind)
  {
    isFloatingWindow = true;
    currentFrontWindowPtr = FrontWindow();
  }
  else
  {
    isFloatingWindow = false;
    currentFrontWindowPtr = FW_findFrontNonFloatWindow();
    lastFloatingWindowPtr = findLastFloatingWindow();
  }

  if(currentFrontWindowPtr != windowPtr)
  {
    if(isFloatingWindow)
    {
      BringToFront(windowPtr);
    }
    else
    {
      if(frontProcess)
        deactivateWindow(currentFrontWindowPtr);

      if(lastFloatingWindowPtr == NULL)
      {
        BringToFront(windowPtr);
      }
      else
      {
        SendBehind(windowPtr,lastFloatingWindowPtr);
      }

      if(frontProcess)
        activateWindow(windowPtr);
    }
  }
}

// ################################################################## FW_HideWindow

void  FW_HideWindow(WindowPtr windowPtr)
{
  WindowPtr frontFloatWindowPtr, frontNonFloatWindowPtr, windowBehind;

  if(getIsWindowVisible(windowPtr) == false)
    return;

  frontFloatWindowPtr = FrontWindow();
  if(getWindowKind(frontFloatWindowPtr) != kFloaterKind)
    frontFloatWindowPtr = NULL;

  frontNonFloatWindowPtr = FW_findFrontNonFloatWindow();

  ShowHide(windowPtr,false);
```

```
      if(windowPtr == frontFloatWindowPtr)
      {
        windowBehind = getNextVisibleWindow(windowPtr);

        if((windowBehind != NULL) && (getWindowKind(windowBehind) == kFloaterKind))
          BringToFront(windowBehind);
      }
      else if(windowPtr == frontNonFloatWindowPtr)
      {
        windowBehind = getNextVisibleWindow(windowPtr);

        if(windowBehind != NULL)
        {
          SendBehind(windowPtr,windowBehind);

          if(isFrontProcess())
            activateWindow(windowBehind);
        }
      }
    }
}

// ###################################################################### FW_ShowWindow

void  FW_ShowWindow(WindowPtr windowPtr)
{
  Boolean                 frontProcess, needsActivate, highlightState;
  Boolean                 modalUp, showingModal;
  SInt16                  windowKind;
  WindowPtr               windowBehind;
  WindowPtr               frontNonFloatWindowPtr;
  ActivateHandlerProcPtr  activateHandler;

  if(getIsWindowVisible(windowPtr) != false)
    return;

  frontProcess = isFrontProcess();

  windowKind = getWindowKind(windowPtr);

  modalUp = isWindowModal(FrontWindow());

  showingModal = isWindowModal(windowPtr);
  if(showingModal)
    BringToFront(windowPtr);

  FW_validateWindowList();

  if(!frontProcess && windowKind == kFloaterKind)
  {
    setWasVisible(windowPtr,true);
    return;
  }

  if(frontProcess)
  {
    if(showingModal)
    {
      highlightState = true;
      needsActivate = true;
    }
    else if(modalUp)
    {
      highlightState = false;
      needsActivate = false;
    }
    else
    {
      if(windowKind == kFloaterKind)
      {
        highlightState = true;
        needsActivate = true;
      }
      else
      {
        frontNonFloatWindowPtr = FW_findFrontNonFloatWindow();
```

```
            if(frontNonFloatWindowPtr == NULL)
            {
              highlightState = true;
              needsActivate = true;
            }
            else
            {
              highlightState = false;
              needsActivate = false;

              windowBehind = getNextWindow(windowPtr);
              while(windowBehind != NULL)
              {
                if(windowBehind == frontNonFloatWindowPtr)
                {
                  deactivateWindow(frontNonFloatWindowPtr);
                  highlightState = true;
                  needsActivate = true;
                  break;
                }
                windowBehind = getNextVisibleWindow(windowBehind);
              }
            }
          }
        }
      }
      else
      {
        highlightState = false;
        needsActivate = false;
      }

      HiliteWindow(windowPtr,highlightState);
      ShowHide(windowPtr,true);

      if(needsActivate)
      {
        activateHandler = getActivateHandler(windowPtr);
        if(activateHandler != NULL)
          CallActivateHandler(activateHandler,windowPtr,true);
      }
    }

// ################################################################### FW_DragWindow

void  FW_DragWindow(WindowPtr windowPtr,Point startPoint,const Rect *draggingBounds)
{
    SInt16    topLimit, newHorizontalWindowPosition, newVerticalWindowPosition;
    Rect      dragRect;
    SInt16    horizontalOffset, verticalOffset;
    GrafPtr   savePort, windowManagerPort;
    KeyMap    keyMap;
    Boolean   commandKeyDown = false;
    RgnHandle dragRegionHdl, windowContentRegionHdl;
    SInt32    dragResult;

    if(StillDown())
    {
      topLimit = LMGetMBarHeight() + 4;
      dragRect = *draggingBounds;
      if(dragRect.top < topLimit)
        dragRect.top = topLimit;

      GetPort(&savePort);
      GetWMgrPort(&windowManagerPort);
      SetPort(windowManagerPort);

      SetClip(GetGrayRgn());

      GetKeys(keyMap);
      if(keyMap[1] & 0x8000)
        commandKeyDown = true;

      if((commandKeyDown == true) || (getWindowKind(windowPtr) != kFloaterKind))
      {
        if(commandKeyDown == false)
```

```
              ClipAbove(FW_findFrontNonFloatWindow());
            else
              ClipAbove(windowPtr);
        }

        dragRegionHdl = NewRgn();
        CopyRgn(getStructureRegion(windowPtr),dragRegionHdl);

        dragResult = DragGrayRgn(dragRegionHdl,startPoint,&dragRect,&dragRect,noConstraint,
                                 NULL);
        SetPort(savePort);

        if(dragResult != 0)
        {
          horizontalOffset = dragResult & 0xFFFF;
          verticalOffset = dragResult >> 16;

          if(verticalOffset != -32768)
          {
            windowContentRegionHdl = NewRgn();
            CopyRgn(getContentRegion(windowPtr),windowContentRegionHdl);

            newHorizontalWindowPosition = (**windowContentRegionHdl).rgnBBox.left +
                                          horizontalOffset;
            newVerticalWindowPosition = (**windowContentRegionHdl).rgnBBox.top +
                                        verticalOffset;
            MoveWindow(windowPtr,newHorizontalWindowPosition,newVerticalWindowPosition,
                       false);

            DisposeRgn(windowContentRegionHdl);
          }
        }

        if(commandKeyDown == false)
          FW_SelectWindow(windowPtr);

        DisposeRgn(dragRegionHdl);
    }
}

// ################################################################ FW_doSuspendEvent

void  FW_doSuspendEvent(void)
{
  WindowPtr currentWindowPtr;
  Boolean   windowIsVisible;

  currentWindowPtr = LMGetWindowList();

  if(isWindowModal(FrontWindow()))
    return;

  while((currentWindowPtr != NULL) &&
        (getWindowKind(currentWindowPtr) == kFloaterKind))
  {
    windowIsVisible = getIsWindowVisible(currentWindowPtr);
    setWasVisible(currentWindowPtr,windowIsVisible);

    if(windowIsVisible)
      ShowHide(currentWindowPtr,false);

    currentWindowPtr = getNextWindow(currentWindowPtr);
  }

  currentWindowPtr = FW_findFrontNonFloatWindow();
  if(currentWindowPtr != NULL)
    deactivateWindow(currentWindowPtr);
}

// ################################################################ FW_doResumeEvent

void  FW_doResumeEvent(void)
{
  WindowPtr currentWindowPtr;
  Boolean   windowWasVisible;
```

```
  currentWindowPtr = LMGetWindowList();

  if(isWindowModal(FrontWindow()))
    return;

  while((currentWindowPtr != NULL) &&
        (getWindowKind(currentWindowPtr) == kFloaterKind))
  {
    windowWasVisible = getWasVisible(currentWindowPtr);
    if(windowWasVisible)
    {
      ShowHide(currentWindowPtr,true);
      activateWindow(currentWindowPtr);
    }

    currentWindowPtr = getNextWindow(currentWindowPtr);
  }

  currentWindowPtr = FW_findFrontNonFloatWindow();
  if(currentWindowPtr != NULL)
    activateWindow(currentWindowPtr);
}

// ################################################### FW_activateFloatsAndFirstDocWin

void  FW_activateFloatsAndFirstDocWin(void)
{
  WindowPtr firstWindowPtr, secondDocumentWindowPtr, currentWindowPtr;
  Boolean   frontProcess;

  if(isWindowModal(FrontWindow()))
    return;

  frontProcess = isFrontProcess();

  if(frontProcess == false)
    FW_doSuspendEvent();
  else
  {
    firstWindowPtr = FrontWindow();
    secondDocumentWindowPtr = FW_findFrontNonFloatWindow();

    if(secondDocumentWindowPtr != NULL)
      secondDocumentWindowPtr = getNextWindow(secondDocumentWindowPtr);

    currentWindowPtr = firstWindowPtr;

    while(currentWindowPtr != secondDocumentWindowPtr)
    {
      if(getIsWindowVisible(currentWindowPtr))
        activateWindow(currentWindowPtr);
      currentWindowPtr = getNextWindow(currentWindowPtr);
    }
  }
}

// ################################################### FW_deactivateFloatsAndFirstDocWin

void  FW_deactivateFloatsAndFirstDocWin(void)
{
  WindowPtr firstWindowPtr, secondDocumentWindowPtr, currentWindowPtr;

  FW_validateWindowList();

  firstWindowPtr = FrontWindow();
  secondDocumentWindowPtr = FW_findFrontNonFloatWindow();

  if(secondDocumentWindowPtr != NULL)
    secondDocumentWindowPtr = getNextWindow(secondDocumentWindowPtr);

  currentWindowPtr = firstWindowPtr;

  while(currentWindowPtr != secondDocumentWindowPtr)
  {
    if(getIsWindowVisible(currentWindowPtr))
      deactivateWindow(currentWindowPtr);
```

```
      currentWindowPtr = getNextWindow(currentWindowPtr);
  }
}

// ########################################################## FW_findFrontNonFloatWindow

WindowPtr  FW_findFrontNonFloatWindow(void)
{
  WindowPtr windowPtr;

  windowPtr = FrontWindow();

  while((windowPtr != NULL) && (getWindowKind(windowPtr) == kFloaterKind))
  {
    do
    {
      windowPtr = getNextWindow(windowPtr);
    } while((windowPtr != NULL) && (getIsWindowVisible(windowPtr) == false));
  }

  return windowPtr;
}

// ############################################################## FW_validateWindowList

void  FW_validateWindowList(void)
{
  WindowPtr windowPtr;
  WindowPtr lastFloatingWindowPtr;

  windowPtr = LMGetWindowList();
  lastFloatingWindowPtr = findLastFloatingWindow();

  if(lastFloatingWindowPtr == NULL)
    return;

  if(getWindowKind(windowPtr) == kFloaterKind)
    return;

  if(isWindowModal(windowPtr))
    return;

  bringFloatersToFront(windowPtr);
}

// #################################################################### FW_GetWRefCon

SInt32  FW_GetWRefCon(WindowPtr windowPtr)
{
  refConExtendHandle  refConExtendHdl;

  refConExtendHdl =(refConExtendHandle) GetWRefCon(windowPtr);
  if(refConExtendHdl != NULL)
    return((*refConExtendHdl)->replacementRefCon);
  else
    return 0;
}

// #################################################################### FW_SetWRefCon

void  FW_SetWRefCon(WindowPtr windowPtr,SInt32 refCon)
{
  refConExtendHandle  refConExtendHdl;

  refConExtendHdl =(refConExtendHandle) GetWRefCon(windowPtr);
  if(refConExtendHdl != NULL)
  {
    (*refConExtendHdl)->replacementRefCon = refCon;
  }
}

// #################################################################### activateWindow

void  activateWindow(WindowPtr windowPtr)
{
  highlightAndActivateWindow(windowPtr,true);
```

```
}

// #################################################################### deactivateWindow

void   deactivateWindow(WindowPtr windowPtr)
{
  highlightAndActivateWindow(windowPtr,false);
}

// ######################################################### highlightAndActivateWindow

void   highlightAndActivateWindow(WindowPtr windowPtr,Boolean activate)
{
  ActivateHandlerProcPtr   activateHandler;

  if(windowPtr == NULL)
    return;

  activateHandler = getActivateHandler(windowPtr);

  HiliteWindow(windowPtr,activate);

  if(activateHandler != NULL)
    CallActivateHandler(activateHandler,windowPtr,activate);
}

// ############################################################ findLastFloatingWindow

WindowPtr   findLastFloatingWindow(void)
{
  WindowPtr windowPtr;

  WindowPtr lastFloatingWindowPtr;

  windowPtr = LMGetWindowList();
  lastFloatingWindowPtr = NULL;

  while(windowPtr != NULL)
  {
    if(getWindowKind(windowPtr) == kFloaterKind)
      lastFloatingWindowPtr = windowPtr;

    windowPtr = getNextWindow(windowPtr);
  }

  return lastFloatingWindowPtr;
}

// ############################################################## checkWindowOrdering

OSErr   checkWindowOrdering(Boolean isFloater,WindowPtr *behind)
{
  OSErr      result = noErr;
  WindowPtr lastFloaterPtr;

  if(isFloater)
  {
    if(((*behind == NULL) && (FW_findFrontNonFloatWindow() != NULL)) ||
       ((*behind != (WindowPtr) -1) && (getWindowKind(*behind) != kFloaterKind)) )
    {
      result = eInvalidWindowOrderingError;
    }
  }
  else
  {
    lastFloaterPtr = findLastFloatingWindow();

    if(lastFloaterPtr != NULL)
    {
      if(*behind == (WindowPtr) -1)
      {
        *behind = lastFloaterPtr;
      }
      else if(*behind != NULL && *behind != lastFloaterPtr &&
              (getWindowKind(*behind) == kFloaterKind))
      {
```

```
            result = eInvalidWindowOrderingError;
        }
      }
    }

    return result;
}

// ############################################################### getNextVisibleWindow

WindowPtr  getNextVisibleWindow(WindowPtr windowPtr)
{
  WindowPtr nextWindowPtr;

  nextWindowPtr = getNextWindow(windowPtr);

  while(nextWindowPtr != NULL)
  {
    if(getIsWindowVisible(nextWindowPtr))
      break;
    nextWindowPtr = getNextWindow(nextWindowPtr);
  }

  return nextWindowPtr;
}

// #################################################################### getWasVisible

Boolean  getWasVisible(WindowPtr windowPtr)
{
  refConExtendHandle  refConExtendHdl;

  refConExtendHdl =(refConExtendHandle) GetWRefCon(windowPtr);

  if(refConExtendHdl != NULL)
    return((*refConExtendHdl)->wasVisible);
  else
    return false;
}

// #################################################################### setWasVisible

void  setWasVisible(WindowPtr windowPtr,Boolean wasVisible)
{
  refConExtendHandle  refConExtendHdl;

  refConExtendHdl = (refConExtendHandle) GetWRefCon(windowPtr);

  if(refConExtendHdl != NULL)
    (*refConExtendHdl)->wasVisible = wasVisible;
}

// ##################################################################### isWindowModal

Boolean  isWindowModal(WindowPtr windowPtr)
{
  SInt16  windowVariant;
  SInt16  windowKind;

  windowVariant = GetWVariant(windowPtr);
  windowKind = getWindowKind(windowPtr);

  if(windowKind == kDialogWindowKind && (windowVariant == 2 || windowVariant == 3))
    return true;
  else
    return false;
}

// ############################################################## bringFloatersToFront

void  bringFloatersToFront(WindowPtr windowPtr)
{
  WindowPtr thisFloaterPtr;

  while(windowPtr != NULL && getWindowKind(windowPtr) != kFloaterKind)
    windowPtr = getNextWindow(windowPtr);
```

```
  if(windowPtr == NULL)
    return;

  thisFloaterPtr = windowPtr;

  bringFloatersToFront(getNextWindow(windowPtr));

  BringToFront(thisFloaterPtr);
}

// ######################################################################## isFrontProcess

Boolean isFrontProcess(void)
{
  ProcessSerialNumber frontPSN, currentPSN;
  OSErr               getFrontProcessResult, getCurrentProcessResult;
  Boolean             isSameProcess = false;

  getFrontProcessResult = GetFrontProcess(&frontPSN);
  getCurrentProcessResult = GetCurrentProcess(&currentPSN);

  if((getFrontProcessResult == noErr) && (getCurrentProcessResult == noErr))
    SameProcess(&frontPSN,&currentPSN,&isSameProcess);

  return(isSameProcess);
}

// ####################################################################### getIsWindowVisible

Boolean  getIsWindowVisible(WindowPtr windowPtr)
{
  return(((WindowPeek) windowPtr)->visible);
}

// ######################################################################### getNextWindow

WindowPtr  getNextWindow(WindowPtr windowPtr)
{
  return((WindowPtr) ((WindowPeek) windowPtr)->nextWindow);
}

// ######################################################################### getWindowKind

SInt16  getWindowKind(WindowPtr windowPtr)
{
  return(((WindowPeek) windowPtr)->windowKind);
}

// ####################################################################### getStructureRegion

RgnHandle  getStructureRegion(WindowPtr windowPtr)
{
  return(((WindowPeek) windowPtr)->strucRgn);
}

// ######################################################################### getContentRegion

RgnHandle  getContentRegion(WindowPtr windowPtr)
{
  return(((WindowPeek) windowPtr)->contRgn);
}

// ######################################################################## getActivateHandler

ActivateHandlerProcPtr  getActivateHandler(WindowPtr windowPtr)
{
  refConExtendHandle  refConExtendHdl;

  refConExtendHdl = (refConExtendHandle) GetWRefCon(windowPtr);
  if(refConExtendHdl != NULL)
    return((*refConExtendHdl)->activateHandler);
  else
    return NULL;
}
```

```
// ############################################################## setActivateHandler

void  setActivateHandler(WindowPtr windowPtr,ActivateHandlerProcPtr activateHandler)
{
  refConExtendHandle  refConExtendHdl;

  refConExtendHdl = (refConExtendHandle) GetWRefCon(windowPtr);

  if(refConExtendHdl != NULL)
    (*refConExtendHdl)->activateHandler = activateHandler;
}

// ################################################################################
```

Demonstration Program Comments

> With regard to the source code file FloatingLib.c, which contains all the special
> functions required to implement floating windows, two options are available. Firstly,
> FloatingLib.c may be added to the FloatingWindows.µ project, along with the source code
> file FloatingWindows.c, in the normal way. Secondly, FloatingLib.c may be compiled as a
> library (68K or PowerPC, as required) and added to the FloatingWindows.µ project as such.
> The latter option has been adopted in the accompanying demonstration program package.

When this program is run, the user should open and close document windows, hide and show the
floating windows (using the Floater menu items and the windows' close boxes), open and close
the About FloatingWindows… modal alert box, open and close the Find movable modal dialog box
(using the Find item in the Edit menu), open and close the Standard File open dialog, and
send/bring the application to the background/foreground. In particular, the user should
observe the following:

•   Document window behaviour, in terms of activation and deactivation, is identical to that
    observed in a normal window environment, including when:

    •   The program is sent to the background and brought to the foreground.

    •   A document window is closed or a new document window is opened.

    •   The About FloatingWindows… modal alert box, the Find… movable modal dialog box, or
        the Standard File open dialog is invoked.

    •   An inactive document window is dragged with the Command key held down.

•   Floating window behaviour is as follows:

    •   The floating window frames are drawn in the inactive state, and their content is
        dimmed, when the alert and dialog boxes are invoked.

    •   The floating windows are hidden when the application is sent to the background and
        shown again when the application is brought to the foreground.

•   When document windows are clicked, the icon at the top left of the window is drawn in
    such a way (normal or dimmed) as to prove that the application "knows" which document
    windows to activate and deactivate.

FloatingWindows.c

#define

rDocumentWindow and the following two constants represent the resource IDs of the 'WIND'
resources for document windows, the "Colours" floating window, and the "Tools" floating window.
The next four constants represent the resource IDs of four 'PICT' resources used by the
floating windows.

kFloaterKind will be assigned to the windowKind field of the floating window window structures.
The next two constants represent error codes that may be returned by a call to the window
creation function in FloatingLib.c.

#typedef

A document structure of type docStruc will be created for each document window. For the purposes of this demonstration, this is simply to prove the correct operation of the two substitute functions FW_SetWRefCon and FW_GetWRefCon. (These functions will assign/retrieve the handle to the window's document structure to/from a reference constant field in a special structure created by the window creation function in FloatingLib.c.)

Global Variables

gColoursFloaterPtr and gToolsFloaterPtr will be assigned pointers to the colour graphics port structures for the floating windows. gIconSuiteHdl will be assigned a handle to an icon suite.

main

Following the call to RegisterAppearanceClient, routine descriptors are created for the three window activation functions.

The call to GetIconSuite creates an icon family. The appropriate icon from this family will be drawn in the top left corner of document windows. The handle will be assigned to the single field in the document window's document structure.

The next two lines call application-defined functions which open one document window and both floating windows.

doOpenDocumentWindow

doOpenDocumentWindow opens a document window.

Firstly, FW_ validateWindowList is called as a safety measure to ensure that all floating windows are in front of all document windows (if any).

FW_GetNewCWindow is called to create the window. The universal procedure pointer to the document window activation function is passed in the fourth parameter. false is passed in the fifth parameter to indicate to FW_GetNewCWindow that this is not a floating window.

If FW_GetNewCWindow does not return an error, a block for a document structure is created and the handle is assigned by FW_SetWRefCon to the reference constant field of a structure created by FW_GetNewCWindow. (FW_GetNewCWindow assigns the handle to this latter structure to the window's refCon field.) The handle to the icon family is assigned to the single field of the document structure.

FW_ShowWindow is called to show the window.

If FW_GetNewCWindow does return an error, an application-defined function is called to handle the error.

doOpenFloatingWindows

doOpenFloatingWindows opens both floating windows.

For each of the two floating windows, FW_GetNewCWindow is called to create the window. A pointer to the appropriate window activation function is passed in the fourth parameter. true is passed in the fifth parameter to indicate to FW_GetNewCWindow that this is a floating window.

If FW_GetNewCWindow does not return an error, GetPicture loads the specified 'PICT' resource, SetWindowPic stores the handle to the picture structure in the windowPic field of the window structure (meaning that the Window Manager will draw the picture in the window instead of generating update events for it), and FW_ShowWindow is called to display the floating window.

If FW_GetNewCWindow does return an error, an application-defined function is called to handle the error.

doEvents

Note that activate events are ignored in the main event loop area because, in a floating windows environment, the normal windows activation/deactivation mechanism must be over-ruled.

At the suspendResumeMessage case in the osEvt case, FW_doResumeEvent is called for a resume event and FW_doResumeEvent is called for a suspend event.

doMouseDown

At the inContent case, if the window in which the mouse-down occurred is not the front non-floating window, FW_SelectWindow is called.

At the inDrag case, FW_DragWindow is called.

At the inGoAway case, if the window is one of the floating windows, FW_HideWindow is called to hide the window. If the window is a document window, FW_DisposeWindow is called to dispose of the window.

doUpdate

An update event results in a call to doDrawWindowContent, which draws an icon in the window in the appropriate state (normal or dimmed). (Recall that, because of the setWindowPic calls in the function doOpenFloatingWindows, the floating windows will not receive update events. Accordingly, the only windows redrawn by the application in this demonstration are the document windows.)

doActivateWindow

doActivateWindow is called from functions within FloatingLib.c. It is not called from within FloatingWindows.c.

doActivateWindow is the window activation function for document windows. doDrawWindowContent is called to draw an icon in the window in the state (normal or dimmed) specified by the becomingActive parameter.

doActivateColoursFloater and doActivateToolsFloater

doActivateColoursFloater and doActivateToolsFloater are called from functions within FloatingLib.c. They are not called from within FloatingWindows.c.

These two functions are the window activation functions for the floating windows. Depending on whether the window is being activated or deactivated, the appropriate picture (normal or dimmed) is drawn in the window and SetWindowPict is called to store the handle to the picture structure in the windowPic field of the window structure (meaning that the Window Manager will draw the picture in the window instead of generating update events for it).

doAdjustMenus

FW_findFrontNonFloatWindow() is called to determine whether any document windows are currently open. If not, the Close item in the File menu is disabled.

If the floating windows are currently hidden, the relevant item in the Floaters menu is unchecked, otherwise it is checked.

doFloatersMenu

doFloatersMenu further processes Floater menu choices.

If the floating window relating to the chosen menu item is currently showing, FW_HideWindow is called to hide the window, otherwise FW_ShowWindow is called to show the window.

doAboutBox, doStandardFileOpen, and doOpenFindDialog

doAboutBox is called when the user chooses the About FloatingWindows… item in the Apple menu. FW_deactivateFloatsAndFirstDocWin is called to deactivate all the document and floating windows before Alert is called to display the About alert box. When the alert box is dismissed, FW_activateFloatsAndFirstDocWin is called to return the windows to the activation status they were in prior to the call to FW_deactivateFloatsAndFirstDocWin.

The same activation/deactivation procedure is followed in the functions doStandardFileOpen and doOpenFindDialog.

doCloseDocWindow

doCloseDocWindow is called in response to a choice of the Close item in the File menu and to a click in a document window's close box.

FW_validateWindowList is called as a safety measure to ensure that all floating windows are in front of all document windows.

The call to FW_findFrontNonFloatWindow returns a pointer to the front document window (and also confirms that at least one document window is open. FW_GetWRefCon is called to get the handle to the window's document structure, which allows that block to be disposed of. FW_DisposeWindow is then called to dispose of the block containing the special data structure created by FW_GetNewCWindow, to remove the window from the screen and window list, and to discard all the window's data storage.

doDrawWindowContent

doDrawDocWindowContent is called when an update event is received and when the document window activate function is called from within FloatingLib.c. It simply draws an icon in the top left of the window in the state (normal or dimmed) specified by the isActive formal parameter.

FloatingLib.c

FloatingLib.c contains the special functions required to support a floating windows environment, including functions which are called in lieu of the usual calls to GetNewCWindow, DisposeWindow, SelectWindow, HideWindow, ShowWindow, DragWindow, SetWRefCon, and GetWRefCon.

#define

kFloaterKind will be assigned to the windowKind field of the window structures for floating windows. The next two constants represent error codes returned by FW_GetNewCWindow. CallActivateHandler simply makes the call to the appropriate window activation handler at the bottom of the functions FW_ShowWindow and highlightAndActivateWindow a little less cryptic.

#typedef

A structure of type refConExtend will be created for all windows, and a handle to that structure will be assigned to the window structure's refCon field. A pointer to the relevant window activation function will be assigned to the first field. The second field will be used to keep track of the window's visibility status. The third field may be used by the application for those purposes for which it would normally use the window structure's refCon field.

FW_GetNewCWindow

To create a window, FW_GetNewCWindow is called in lieu of the usual call to GetNewCWindow. The isFloater parameter specifies whether the window is to be a document window or a floating window.

At the first line, failure is assumed.

checkWindowOrdering is called to ensure that the behind formal parameter is not attempting to open the window at an inappropriate "behind" or "in front of" location in the window list. As part of this process, and in the case of document windows, checkWindowOrdering changes a behind assignment of (WindowPtr) -1 to the pointer to the last floating window, causing the window to be opened behind that floating window rather than in front of all windows. Apart from that particular case, inappropriate behind specifications will cause an error to be returned.

The call to NewHandle creates a block for a refConExtend structure. If this call is not successful, the function exits and an error code is returned to the calling function.

The call to GetNewCWindow creates the window, following which a pointer to the window is returned to the calling function. The next three lines assign the handle to the refConExtend structure to the window structure's refCon field and assign the previous value in the refCon field to the refConExtend structure's replacementRefCon field. The next line assigns the pointer to the window's activation function to the refConExtend structure's activateHandler field.

If the visible field of the window's window structure indicates that the window is currently visible, and if the application is currently not the front process, ShowHide is called to hide the window and setWasVisible is called to record the visibility status of the window by assigning true to the refConExtend structure's wasVisible field. (This block caters for the possibility of windows being created while the application is in the background, for example, by Apple events. It prevents the window from being highlighted on creation.)

If the call to GetNewCWindow was not successful, the subsequent if block is bypassed, DisposeHandle is called to dispose of the refConExtend structure, and an error code is returned to the calling function.

FW_DisposeWindow

FW_DisposeWindow is called in lieu of the usual call to DisposeWindow.

If the specified window (which will invariably be a document window) is visible, a call is made to FW_HideWindow so that the next document window in the list (if any) is activated. The window's refConExtend structure is then disposed of, following which DisposeWindow is called to remove the window from the screen, remove it from the window list, and discard all its data storage.

FW_SelectWindow

FW_SelectWindow is called in lieu of the usual call to SelectWindow. It selects the window and brings it to the front of its section of the window list. Selecting a floating window makes it the absolute frontmost window on the screen, whereas selecting a document window makes it the frontmost window behind the floating windows (or, if no floating windows are open, the absolute frontmost window).

The first line determines whether the application is the front process.

If the window clicked is of the floating kind, that fact is recorded and the pointer to the current front floating window is retrieved. Otherwise, if the window clicked is a document window, that fact is recorded, and pointers to the first document window and last floating window in the window list are retrieved.

If the window clicked is not the current front window in either the floating or document window sections of the window list and if the window clicked is a floating window, that window is brought to the very front of the list by the call to BringToFront. Otherwise, if the window clicked is a document window: if the application is the front process, the current front document window is deactivated; if their are no floating windows, BringToFront is called to bring the window to the front; if there are floating windows, SendBehind is called to locate the window immediately behind the last floating window; if the application is the front process, the window is activated.

FW_HideWindow

FW_HideWindow is called in lieu of the usual call to HideWindow. If the window is the frontmost window, it is moved behind the window immediately behind it.

At the first two lines, if the window is not visible, the function returns without doing anything.

The next line gets the pointer to the frontmost window. If this window is not a floating window, a variable is set to record that fact.

The next line gets the pointer to the first document window.

The call to ShowHide hides the specified window without affecting the front-to-back ordering of the windows.

If the newly hidden window is the front floating window, and if the next window in the list (if any) is a floating window, BringToFront is called to bring that next window to the front. Otherwise, if the newly hidden window is the first document window, and if there is another visible document window, SendBehind is called to move the newly hidden window behind that visible document window. Also, if the application is the front process, that visible document window is activated.

FW_ShowWindow

FW_ShowWindow is called in lieu of the usual call to ShowWindow. If the specified (hidden) window is the frontmost document window, the window behind it is deactivated and the specified window is activated. If the window is modal, the function makes sure it is in front.

At the first two lines, if the specified window is currently visible, the function returns without doing anything. The next line determines whether the application is the front process.

The next line records whether the specified window is a floating window. The line after that records whether the front window is a modal window.

At the next block, if the specified window is a modal window, BringToFront is called to bring it to the front (without activating it).

Before proceeding further, FW_validateWindowList is called as a safety measure to ensure that all windows in the window list are in the right place.

If the application is not the front process, and the specified window is a floating window, it is marked as visible so that it will be shown on resume. (The function setWasVisible assigns true to the wasVisible field of the window's refConExtend structure.)

The large if block executes only if the application is the front process. If the specified window is modal, two local variables are set to true (ensuring that modal windows will always be activated and highlighted). If the front window is modal, these variables are set to false (ensuring that, whatever the specified window type, it will not be activated and highlighted when a modal window is up). If neither the specified nor front window is modal:

• If the specified window is a floating window, the two local variables are set to true. (New floating windows are always highlighted and activated.)

• If the specified window is a document window, and it is the only document window, the two local variables are set to true so that it will be highlighted and activated. If it is not the only document window: the two local variables are initially set to false (assume the specified window is behind the front one); the window list is walked to determine whether the specified window is coming up in front of the frontmost document window, in which case the front document window is deactivated and the variables are set so that the specified window will be highlighted and activated.

If the application is not the front process (the last else block), the two local variables are set to false so that no highlighting/activation will occur.

HiliteState is called to either unhighlight or highlight the specified window depending on the result of the previous machinations. If the previous machinations determined that the window should be activated, pointer to the window's activation function is retrieved from the window's refConExtend structure and passed in a call to CallActivateHandlerProc, which causes the activation function to execute.

FW_DragWindow

FW_DragWindow is called in lieu of the usual call to DragWindow. It drags the specified window around, ensuring that document windows remain behind floating windows. Like DragWindow, FW_DragWindow does not bring the window forward if the Command key is held down during the drag.

WaitMouseUp tests whether the mouse button has remained down since the last mouse-down event. If it has, the following occurs.

The first block adjusts the top of the dragging rectangle so that it is below the menu bar. The next block saves the current colour graphics port and sets the window manager port as the current colour graphics port. The call to SetClip sets the clipping region to the region below the menu bar.

The next three lines check whether the Command key is down and, if so, set a variable to record that fact.

If the Command key is not down and the window is not a floating window, the ClipAbove call at sets the clipping region to the gray region minus the structure regions of all windows in front of the front non-floating window. (In this instance, the front document window is being dragged, so the windows in front are the floating windows.) If the window is not a floating window and the Command key is down, the ClipAbove call sets the clipping region to the gray region minus the structure regions of all windows in front of the window being dragged. (In this instance, there could be one or more document windows, as well as floating windows, above the document window being dragged.)

The next two lines create a region to drag, specifically, the structure region of the specified window. This is passed as a parameter to the call to DragGrayRgn, which moves a dotted outline of the region, following the mouse as it moves and retaining control until the mouse button is released.

When the mouse button is released, SetPort sets the colour graphics port to the saved earlier.

DragGrayRgn returns an SInt32. If the mouse was outside the slop rectangle when the button was released, -32768 is returned in both words, otherwise the high word contains the vertical distance moved and the low word contains the horizontal distance moved. If the value returned is not zero, the value in both words is retrieved. If the mouse was not outside the slop rectangle, the new horizontal and vertical global coordinates are calculated and passed as a parameter in the MoveWindow call, which moves the window to the new location without bringing it to the front.

If the Command key was not down during the drag (the third last line), the call to
FW_SelectWindow brings the window to the absolute front of the window list (floating window) or
to the front of the document windows section of the list (document window).

FW_doSuspendEvent

FW_doSuspendEvent hides any visible floating windows and deactivates the frontmost document
window. It should be called when the application receives a suspend event.

The first line gets the pointer to the front window's window structure from the low-memory
global WindowList.

If the first visible window is modal, the floating and front document windows will already be
deactivated, so the function simply returns.

The while loop executes for all floating windows. The current visibility status of the window
is saved to the wasVisible field of its window's refConExtend structure. If the window is
visible, it is hidden without affecting the front-to-back ordering of the open windows.

At the last block, if there are any document windows, the frontmost document window is
deactivated.

FW_doResumeEvent

FW_doResumeEvent shows all floating windows which were hidden by FW_doSuspendEvent. As each
window is shown, the activation function for that window is called. This function should be
called when the application receives a resume event.

The first line gets the pointer to the front window's window structure from the low-memory
global WindowList.

If the first visible window is modal, the floating and front document windows will already be
activated, so the function simply returns.

The while loop executes for all floating windows. The saved visibility status of the window is
retrieved from the wasVisible field of its window's refConExtend structure. If the window was
visible when FW_doSuspendEvent was called, it is shown without affecting the front-to-back
ordering of the open windows.

At the last block, if there are any document windows, the frontmost document window is
activated.

FW_activateFloatsAndFirstDocWin

FW_activateFloatsAndFirstDocWin activates those windows which were visible and activated before
deactivateFloatersAndFirstDocWin was called. It should be called immediately after an modal
alert or dialog box is dismissed.

The first two lines cause the function to assume that, if a modal dialog is up, the caller did
not really mean to call this function.

The next line determines if the application is the front process.

If the application is in the background, the floating windows should be hidden rather than
shown, so FW_doSuspendEvent is called.

If the application is in the foreground, activateWindow is called on all visible windows in the
list up to but not including the second document window in the list.

FW_deactivateFloatsAndFirstDocWin

FW_deactivateFloatsAndFirstDocWin deactivates all visible floating windows and the active
document window. It should be called immediately before an modal alert or dialog box is
dismissed.

The first line is a safety check to ensure that the window ordering has not been changed.

deactivateWindow is called on all visible windows in the list up to but not including the
second document window in the list.

FW_findFrontNonFloatWindow

FW_findFrontNonFloatWindow returns a pointer to the first visible window in the window list that is not a floating window.

FW_validateWindowList

FW_validateWindowList is called from FW_ShowWindow and FW_deactivateFloatsAndFirstDocWin. It should also be called from the application immediately before document windows are opened and closed. It is intended to cater for the remote possibility that document windows might somehow have been pulled in front of floating windows, in which case it will pull the floating windows to the front.

If there are no floating windows, the function simply returns. If the first window in the window list is a floating window, the function assumes that all is well. (Note that the function does not check whether a document window is "between" the floating windows.) If the first window is modal, the function returns.

If a return has not occurred prior to the last line, that line calls a function which brings the floating windows to the front.

FW_GetWRefCon and FW_SetWRefCon

FW_GetWRefCon and FW_SetWRefCon retrieve/assign values from/to the replacementRefCon field of the window's refConExtend structure.

activateWindow, deactivateWindow, and highlightAndActivateWindow

activateWindow and/or deactivateWindow are called from FW_SelectWindow, FW_HideWindow, FW_ShowWindow, FW_doSuspendEvent, FW_doResumeEvent, FW_deactivateFloatsAndFirstDocWin, and FW_activateFloatsAndFirstDocWin. They simply set a Boolean variable to indicate whether the specified window is to be highlighted and activated or unhighlighted and deactivated, and then pass further processing to highlightAndActivateWindow.

highlightAndActivateWindow highlights or unhighlights the specified window, retrieves the pointer to the window's activation function from the window's refConExtend structure, and calls that function.

findLastFloatingWindow

findLastFloatingWindow returns a pointer to last floating window in the window list.

checkWindowOrdering

checkWindowOrdering is called from FW_GetNewCWindow to ensure that the behind formal parameter is not attempting to open the window at an inappropriate "behind" or "in front of" location in the window list. As part of this process, and in the case of document windows, checkWindowOrdering changes a behind assignment of (WindowPtr) -1 to the pointer to the last floating window, causing the window to be opened behind that floating window rather than in front of all windows. Apart from that particular case, inappropriate behind specifications will cause an error to be returned.

If the window to be created is a floating window, an error will be returned if:

•   The behind formal parameter calls for the window to be created behind all other windows when non-floating windows exist.

•   The behind formal parameter specifies the window behind which the new window is to be created and that window is not a floating window.

If the window to be created is a document window, and if a floating window exists:

•   If the behind formal parameter specifies that the window should be created in front of all other windows, the behind parameter is changed so that the window will be created behind the last floating window.

•   If the behind formal parameter specifies that the window behind which the window is to be opened is a floating window, but it is not the last floating window, an error is returned.

getNextVisibleWindow, getWasVisible, and setWasVisible

getNextVisibleWindow returns the pointer to the next visible window in the window list.

getWasVisible returns the value in the wasVisible field of the specified window's refConExtend structure.

setWasVisible sets the value in the wasVisible field of the specified window's refConExtend structure.

isWindowModal

isWindowModal determines whether a window is modal.

If the value in the window structure's windowKind field is kDialogWindowKind and the window's variant code is 2 or 3 (the variant codes for kWindowModalDialogProc and kWindowMovableModalDialogProc dialog boxes), true is returned.

> This function assumes that an application's modeless dialog boxes will not use kWindowGrowDocumentProc or kWindowVertZoomDocumentProc windows (which also have variant codes of 2 and 3). It also assumes that only kWindowModalDialogProc and kWindowMovableModalDialogProc windows will be used for modal and movable modal dialog boxes. If this is not the case, this function will have to be modified.

bringFloatersToFront

bringFloatersToFront is called from FW_validateWindowList in the unlikely event that the floating windows are not in front of the document windows. This is a recursive function (that is, it calls itself) that brings all floating windows to the front of the window list.

When the while loop finds a floating window, the function remembers it (thisFloaterPtr = windowPtr) and then calls itself to look further. When the end of the window list is reached (thisWindow == NULL), the function "unwinds", calling BringToFront on each found floating window.

isFrontProcess

isFrontProcess compares the serial numbers of the current process and the front process and returns true if they are the same, that is, the application is the front process.

getIsWindowVisible, getNextWindow, getWindowKind, getStructureRegion, and getContentRegion

getIsWindowVisible returns the value in the visible field of the specified window's window structure.

getNextWindow returns the value in the nextWindow field of the specified window's window structure.

getWindowKind returns the value in the windowKind field of the specified window's window structure.

getStructureRegion returns the RgnHandle in the strucRgn field of the specified window's window structure.

getContentRegion returns the RgnHandle in the contRgn field of the specified window's window structure.

getActivateHandler and setActivateHandler

getActivateHandler returns the pointer in the activateHandler field of the specified window's refConExtend structure.

setActivateHandler assigns a pointer to the activateHandler field of the specified window's refConExtend structure.