20 Version 1.1

FLOATING WINDOWS AND CUSTOM WINDOW DEFINITION FUNCTIONS

Includes Demonstration Program Floaters

Floating Windows

Floating windows, which are often referred to in the jargon as **windoids**, 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.

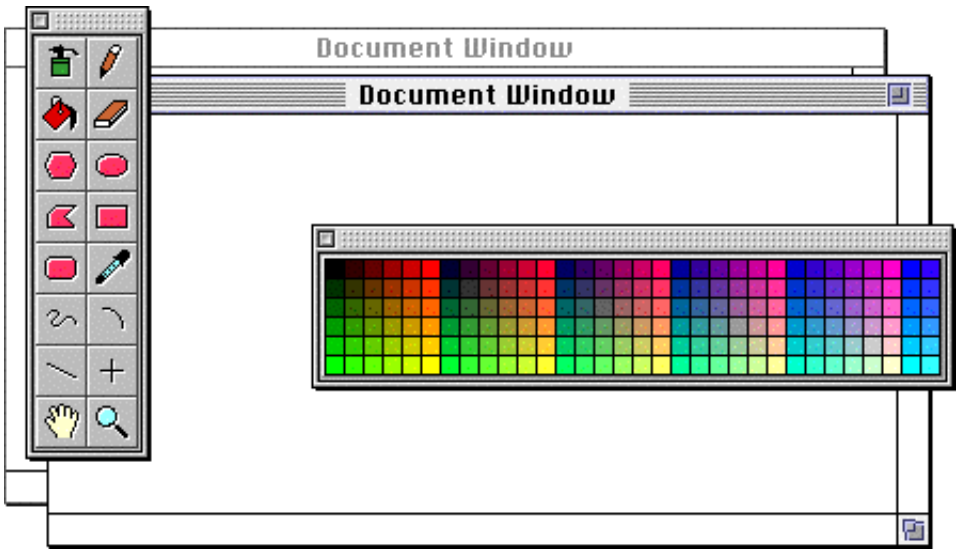


FIG 1 - FLOATING WINDOWS - EXAMPLES

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 dialogs and alerts.
- Floating windows.
- Document windows and modeless dialogs.

Note that floating windows should remain behind modal dialogs and alerts, 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.

Appearance of Floating Windows

The appearance of a floating window is defined by a special **window definition function**. Window definition functions are stored in resources of type `'WDEF'`.

The only published appearance specification applicable to floating windows is that for utility windows in the document Apple Grayscale Appearance for System 7.5 published by Apple Computer, Inc (see Fig 2). As shown at Fig 2, a floating window can have a close box, a zoom box[2], a size box, and a title. The title (if any) should be in the application font, specifically, 10 point Geneva bold. The user should be able to drag the window using any part of the window frame.

FIG 2 - APPEARANCE OF UTILITY WINDOWS AS DEFINED BY APPLE GRAYSCALE APPEARANCE FOR SYSTEM 7.5

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, windows should almost always appear in the active state. The exception is when a modal dialog or alert box is presented to the user. When this occurs, the appaearance of all floating windows should be changed to reflect the inactive state. As a further refinement, the content of the window may 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.

---

[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.

[2]Zoom boxes are sometimes used in floating windows to simply collapse the window to its drag bar and open it out to a fixed size.

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 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 routines, such as `SelectWindow`, `ShowWindow`, and `HideWindow`, which implicitly generate activate and deactivate events. Instead, you must use lower-level routines 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 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.

Conceptually, these considerations require you to subvert the system software's normal window activation/deactivation activities and to 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 Routines

Implementing floating windows in an application basically involves providing a number of application-defined substitute and supporting routines, 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.

Substitute Routines

The substitute routines are routines which should be called in lieu of those Macintosh ToolBox routines that would normally be called in a non-floating windows environment. These substitute routines[3], the requirements of those routines, and the Toolbox calls they replace, are as follows:

[3]The names of the replacement and supplementary routines 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.

Replacement	Replaces	Requirements
newGetNewWindow	GetNewWindow 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.
newDisposeWindow	DisposeWindow	Dispose of document windows. Activate the next document window in the window list, remove the specified document window from the screen and the window list using CloseWindow and dispose of the window's window record.
newSelectWindow	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.
newHideWindow	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.
newShowWindow	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 dialog is present, show the window in the inactive state.
newDragWindow	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.

Supporting Routines

The main supporting routines, and the requirements of those routines, are described in the following:

Routine	Requirements
handleSuspendEvent	Hide any floating windows, and unhighlight and deactivate the frontmost document window. (Call this routine when the application receives a suspend event.)
handleResumeEvent	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 routine when the application receives a suspend event.)
deactivateFloatsAndFirstDocWin	Unhighlight and deactivate any visible floating windows and the frontmost document window. (Call this routine immediately before a modal dialog or alert box is invoked.)
activateFloatsAndFirstDocWin	Highlight and activate those windows which were visible, highlighted and activated before deactivateFloatersAndFirstDocWin was called. However, if the application is in the background when this function is called (such as when a movable modal progress dialog was up and then disappears), do not perform those actions. Instead, simply call handleSuspendEvent to hide any visible floating windows. (Call this routine immediately after an alert or modal dialog box is dismissed.)

Custom Window Definition Functions

If your application defines its own window types, you must supply your own window definition function (WDEF). To create a custom window type, you must write your own WDEF, compile it as a

resource of type 'WDEF', and store it in the resource fork of the application that uses it. Custom 'WDEF' resources must have an ID of 128 to 4096. ('WDEF' resource IDs from 0 to 127 are reserved by Apple.)

When your application creates a window, the WDEF is read into memory and a handle to it is placed in the window record's windowDefProc field. Then, when the Window Manager needs to perform a window type-dependent action on the window, it calls the WDEF to perform that action.

WDEFs are required to:

- Draw the window frame.
- Report the region in which mouse-down events occur.
- Calculate the window's content and structure regions.
- If the window type supports resizing, draw the size box and resize the window frame when the user drags the size box.
- Perform any customised initialisation or disposal tasks.

You must declare your WDEF like this:

```
pascal SInt32  windowDef(short varCode,WindowPtr theWindow,short message,SInt32 param);
```

`varCode` The variation code for the window.

A WDEF can support up to 16 variation codes, identified by integers 0 to 15. To invoke the desired window type, you specify the window's definition ID. To derive the window definition ID for the window, add the variation code value to the result of 16 multiplied by the resource ID of the 'WDEF' resource.

`theWindow` A pointer to the window's window record.

`message` A value which specifies which operation the WDEF must perform. Possible values are as follows:

Constant	Value	Operation
wDraw	0	Draw the window frame.
wHit	1	Determine where a mouse-down event occurred.
wCalcRgns	2	Calculate the content and structure regions.
wNew	3	Perform any required additional initialisation.
wDispose	4	Perform any additional disposal actions.
wGrow	5	Draw the grow image during resizing.
wDrawGIcon	6	Draw the size box and scroll bar outlines.

`param` A value whose meaning depends on the operation specified in the message parameter.

When your WDEF performs the action specified in the message parameter, it must return a function result or, if the action requires no result code, it must return 0.

Responding to message Parameter Values

The following describes how your WDEF should respond to values passed by the Window Manager in the message parameter.

wDraw

Action Required by Window Manager: Draw the window frame in the current graphics port (which, when the WDEF is called, is set by the Window Manager to the **Window Manager port**).

Value in `param` **:** The low-order word[4] contains either `0` (meaning draw the entire window frame) or `wInGoAway` (`4`) (meaning add highlighting to, or remove it from, the window's close box).

When the value in `param` is `0`, and before drawing the window frame, you must make certain checks as follows:

- If the value in the window record's `visible` field is `false`, do nothing.
- If the value in the window record's `hilited` field is `true`:
  - Draw the window frame in such a way as to indicate that the window is active.
  - If the `goAwayFlag` field in the window record is `true`, draw a close box in the window frame.
- If the value in the window record's `hilited` field is `false`, draw the window frame in such a way as to indicate that the window is inactive.

The window frame typically, but not necessarily, includes the window title. In non-utility windows, the title should be displayed in the system font and system font size. (The Window Manager port is already set to use the system font and system font size.) In utility windows, the title should be drawn in the application font, 10 point, bold.

Value to Return: Always return `0`.

`wHit`

Action Required by Window Manager: Determine where the cursor was when the mouse button was pressed.

Value in `param` **:** Mouse location in global coordinates. The vertical coordinate is in the high-order word and the horizontal coordinate is in the low-order word.

Value to Return: One of the following constants:

Constant	Value	Meaning
`wNoHit`	`0`	None of the following.
`wInContent`	`1`	In the content region (except the grow region if the window is active.)
`wInDrag`	`2`	In the drag region.
`wInGrow`	`3`	In the grow region (window active only).
`wInGoAway`	`4`	In the go-away region (window active only).
`wInZoomIn`	`5`	In the zoom box for zooming in (active window only).
`wInZoomOut`	`6`	In the zoom box for zooming out (active window only).

The return value `wNoHit` might mean (but not necessarily) that the point is not in the window. The standard window definition functions, for example, return `wNoHit` if the point is in the window frame but not in the title bar.

`wCalcRgns`

Action Required by Window Manager: Calculate the window's content and structure regions based on its port rectangle.

Value in `param` **:** (Not applicable. Ignore.)

The handles to the content and structure regions are in, respectively, the `contRgn` and `strucRgn` fields of the window record. The regions are in global coordinates.

[4]Note that, in the case of the `wDraw` message, the high-order word may contain undefined data; therefore, evaluate only the low word.

Note that the Window Manager requests this operation only if the window is visible.

Value to Return: Always return `0`.

wNew

Action Required by Window Manager: Perform any window type-specific initialisation.

Value in `param` : (Not applicable. Ignore.)

As an example of type-specific initialisation, the initialisation routine for a standard document window creates the `wStateData` record for storing zooming data.

Value to Return: Always return `0`.

wDispose

Action Required by Window Manager: Perform any additional tasks associated with disposing of a window.

Value in `param` : (Not applicable. Ignore.)

As an example of an additional disposal task, the disposal routine for a standard document window disposes of the `wStateData` record associated with the window.

Value to Return: Always return `0`.

wGrow

Action Required by Window Manager: Draw a grow image of the window.

Value in `param` : Pointer to a rectangle, in global coordinates, whose upper-left corner is aligned with the port rectangle of the window's graphics port.

Your grow image must fit inside the rectangle passed in the `param` parameter.

As the user drags the mouse, the Window Manager repeatedly sends `wGrow` messages, so that you can change your grow image to match the changing mouse location. Draw the grow image in the current graphics port (which is the Window Manager port) in the current pen pattern and mode. (In the Window Manager port, the pen pattern and mode will be set up as `gray` and `notPatXor` to conform to user interface guidelines.)

Note that the grow routine for a standard document window draws a dotted (`gray`) outline of the window and also the lines delimiting the title bar, size box, and scroll bar areas.

Value to Return: Always return `0`.

wDrawGIcon

Action Required by Window Manager: Draw the size box in the content region if the window is active. If the window is inactive, draw whatever is appropriate to show that the window cannot currently be resized.

Value in `param` : (Not applicable. Ignore.)

If the size box is located in the window frame instead of the content region, do nothing. Instead, draw the size box in response to the `wDraw` message.

Note that the draw grow icon routine for an active standard document window draws the size box plus the delimiting lines for the scroll bar areas. For an inactive window, it erases the size box and draws the delimiting lines.

Value to Return: Always return `0`.

Problems With Purgeable Custom WDEF Resources

Like other definition functions, WDEFs contain executable code that needs to be locked down in memory whenever it is executed.

If your application is using a custom WDEF marked as purgeable, the Memory Manager may purge the WDEF resource in order to allocate additional memory in your application heap. The Window Manager will, of course, need to call the WDEF to redraw your windows whenever they are erased. If this requirement arises, the Window Manager will simply reload the WDEF before calling it. Even if your application is not the frontmost application, the Window Manager will be able to reload the WDEF provided your application is the application that is executing at the time, that is, it is the "current" application. In this circumstance, there is no problem.

A problem can arise, however, when your application is not the frontmost application and is not the "current" application. In this circumstance, your application's context has not been restored and your resource chain is not the current resource chain. As a consequence, the Window Manager attempts to load the WDEF from the wrong resource chain. Since it cannot find the WDEF, the result is System Error 87.

This problem was first documented by Apple in late 1996. There are two reasons why the problem has only recently been detected:

- Most applications use custom WDEFs for floating windows, and applications should hide floating windows before being suspended. The Window Manager will therefore never need to call the WDEF to redraw floating windows when the owner application is in the background.
- Applications in the background do not normally allocate large amounts of memory. In addition, most applications are allocated generous memory partitions. Accordingly, the Memory Manager may never need to purge the WDEF in order to satisfy a memory request.

The simple solution to the problem is to invariably mark all custom WDEFs as non-purgeable.

Note that WDEFs in the System file can safely be purged because they are always in the resource chain. Note also that this problem does not apply to other custom definition functions (CDEFs, MDEFs, etc.), the reason being that they are never called when the application's process globals are not current.

Relevant Window Manager Constants and Routines

Constants

Window Definition Function Task Codes

```
wDraw       = 0
wHit        = 1
wCalcRgns   = 2
wNew        = 3
wDispose    = 4
wGrow       = 5
wDrawGIcon  = 6
```

Window Definition Functions `wHit` Return Codes

```
wNoHit      = 0
wInContent  = 1
wInDrag     = 2
```

```
wInGrow     = 3
wInGoAway   = 4
wInZoomIn   = 5
wInZoomOut  = 6
```

Routines

```
void   GetWMgrPort(GrafPtr *wPort);
void   GetCWMgrPort(CGrafPtr *wMgrCPort);
void   ShowHide(WindowRef theWindow,Boolean showFlag);
void   HiliteWindow(WindowRef theWindow,Boolean fHilite);
void   BringToFront(WindowRef theWindow);
void   SendBehind(WindowRef theWindow,WindowPtr behindWindow);
void   MoveWindow(WindowRef theWindow,short hGlobal,short vGlobal,Boolean front);
void   CloseWindow(WindowRef theWindow);
void   ClipAbove(WindowRef window);
long   DragGrayRgn(RgnHandle theRgn,Point startPt,const Rect *limitRect,
       const Rect *slopRect,short axis,DragGrayRgnUPP actionProc);
```

Demonstration Program

```
1  // ##############################################################################################
2  // Floaters.h
3  // ##############################################################################################
4  //
5  // This program opens two floating windows and allows the user to open up to three
6  // document windows.  The floating windows and the document windows may be closed and
7  // opened via their close boxes, by choosing the relevant items in the File and Floaters
8  // menus, and by pressing the relevant Command key equivalents.
9  //
10 // The program is supported by specialised floating windows routines in FloatRoutines.c
11 // and by a custom window definition function which defines the appearance and behaviour
12 // of the floating windows.
13 //
14 // The program utilises the following resources:
15 //
16 // •  An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit and Floaters menus
17 //    (preload, non-purgeable).
18 //
19 // •  'WIND' resources (purgeable) (initially visible) for the document and floating
20 //    windows.
21 //
22 // •  An 'ALRT' resource (purgeable), and associated 'DITL' resource (purgeable), for
23 //    an alert invoked by the user choosing the About… item in the Apple menu.
24 //
25 // •  A 'DLOG' resource (purgeable), and associated 'DITL' resource (purgeable), for a
26 //    dialog box invoked when the user chooses the Find item in the File menu.
27 //
28 // •  'PICT' resources (non-purgeable, colour versions preload) containing pictures to
29 //    be drawn in the floating windows.
30 //
31 // •  A 'snd ' resource (purgeable) containing a sound invoked when the user clicks
32 //    in the content region of the document and floating windows.
33 //
34 // •  A 'STR# ' resource (purgeable) containing text to be drawn in the document windows.
35 //
36 // •  A custom 'WDEF' resource (non-purgeable) containing the window definition function
37 //    utilised by the floating windows.
38 //
39 // •  A 'SIZE' resource with the acceptSuspendResumeEvents and doesActivateOnFGSwitch
40 //    flags set.
41 //
42 // ##############################################################################################
43
44 // …… includes
45
46 #include <Fonts.h>
47 #include <Menus.h>
48 #include <TextEdit.h>
49 #include <Dialogs.h>
50 #include <SegLoad.h>
51 #include <ToolUtils.h>
52 #include <Devices.h>
```

```
#include <Resources.h>
#include <Sound.h>
#include <Gestalt.h>
#include <Processes.h>
#include <LowMem.h>

// ……… defines

#define mApple                128
#define  iAbout               1
#define mFile                 129
#define  iNew                 1
#define  iClose               4
#define  iFind                11
#define  iQuit                13
#define mEdit                 130
#define mFloaters             131
#define  iTools               1
#define  iColours             2
#define rMenubar              128
#define rDocWindow            128
#define rToolsWindow          129
#define rColoursWindow        130
#define rAboutAlert           128
#define rFindDialog           129
#define rToolsPict            128
#define rColoursPict          129
#define rToolsPictDim         130
#define rColoursPictDim       131
#define rSound                8192
#define kDocumentKind         1
#define kFloatingKind         2
#define MAXLONG               0x7FFFFFFF

// ……… typedefs

typedef pascal void (*ActivateProcPtr) (WindowPtr theWindow,Boolean activate);

typedef struct
{
  SInt16          windowType;
  ActivateProcPtr activateHandler;
  Boolean         wasVisible;
} docRecord, **docRecordHandle;

// ……… function prototypes

// FloaterDemo.c

void            main                          (void);
void            doInitManagers                (void);
void            doOpenFloatingWindows         (void);
void            eventLoop                     (void);
void            doEvents                      (EventRecord *);
void            doMouseDown                   (EventRecord *);
void            doUpdate                      (EventRecord *);
void            doActivate                    (EventRecord *);
void            doOSEvent                     (EventRecord *);
void            doAdjustMenus                 (void);
void            doMenuChoice                  (SInt32);
void            doFileMenu                    (SInt16);
void            doFloatersMenu                (SInt16);
void            doOpenDocWindow               (void);
void            doCloseDocWindow              (void);
void            doDrawDocWindowContent        (void);
void            doProvingBeeps                (WindowPtr);
void            doOpenFindDialog              (void);
void            doDisposeFindDialog           (void);
pascal void     docActivateHandler            (WindowPtr,Boolean);
pascal void     toolsActivateHandler          (WindowPtr,Boolean);
pascal void     coloursActivateHandler        (WindowPtr,Boolean);
void            invalidateScrollBarArea       (WindowPtr);

// FloatRoutines.c

WindowPtr       newGetNewWindow               (SInt16,WindowPtr,ActivateProcPtr,SInt16);
void            newDisposeWindow              (WindowPtr);
```

```
void            newSelectWindow                (WindowPtr);
void            newHideWindow                  (WindowPtr);
void            newShowWindow                  (WindowPtr);
void            newDragWindow                  (WindowPtr,Point,const Rect *);
void            handleSuspendEvent             (void);
void            handleResumeEvent              (void);
void            deactivateFloatsAndFirstDocWin (void);
void            activateFloatsAndFirstDocWin   (void);
void            deactivateWindow               (WindowPtr);
void            activateWindow                 (WindowPtr);
void            highlightAndActivateWindow     (WindowPtr,Boolean);
WindowPtr       findFrontNonFloatingWindow     (void);
WindowPtr       findLastFloatingWindow         (void);
Boolean         isWindowModal                  (WindowPtr);
WindowPtr       getWindowList                  (void);
void            setWindowList                  (WindowPtr);
WindowPtr       getNextWindow                  (WindowPtr);
void            setNextWindow                  (WindowPtr,WindowPtr);
Boolean         getWasVisible                  (WindowPtr);
void            setWasVisible                  (WindowPtr,Boolean);
SInt16          getWindowKind                  (WindowPtr);
void            setWindowKind                  (WindowPtr,SInt16);
Boolean         getWindowVisible               (WindowPtr);
RgnHandle       getStructureRegion             (WindowPtr);
RgnHandle       getContentRegion               (WindowPtr);
void            setWindowHilite                (WindowPtr,Boolean);
ActivateProcPtr getActivateHandler             (WindowPtr);
void            setActivateHandler             (WindowPtr,ActivateProcPtr);

// ############################################################################################

// ############################################################################################
// FloaterDemo.c
// ############################################################################################

// ……… includes

#include "Floaters.h"

// ……… global variables

Boolean   gColorQuickDrawPresent = false;
Boolean   gColourDisplay         = false;
Boolean   gDone;
Boolean   gInBackground;
SInt16    gNumberDocWindowsOpen;
WindowPtr gToolsWindowPtr;
WindowPtr gColoursWindowPtr;

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

void  main(void)
{
  OSErr       osErr;
  SInt32      response;
  GDHandle    mainDeviceHdl;
  SInt16      bitsPerPixel;
  Handle      menubarHdl;
  MenuHandle  menuHdl;

  // ……… initialise managers

  doInitManagers();

  // ……… check for Color QuickDraw

  osErr = Gestalt(gestaltQuickdrawVersion,&response);
  if(response >= gestalt8BitQD)
  {
    gColorQuickDrawPresent = true;

    mainDeviceHdl = LMGetMainDevice();
    bitsPerPixel = (*(*mainDeviceHdl)->gdPMap)->pixelSize;
    if(bitsPerPixel > 1)
      gColourDisplay = true;
  }

```

```
207     // ……… set up menu bar and menus
208
209     menubarHdl = GetNewMBar(rMenubar);
210     if(menubarHdl == NULL)
211       ExitToShell();
212     SetMenuBar(menubarHdl);
213     DrawMenuBar();
214
215     menuHdl = GetMenuHandle(mApple);
216     if(menuHdl == NULL)
217       ExitToShell();
218     else
219       AppendResMenu(menuHdl,'DRVR');
220
221     // ……… open floating windows
222
223     doOpenFloatingWindows();
224
225     CheckItem(GetMenuHandle(mFloaters),iTools,true);
226     CheckItem(GetMenuHandle(mFloaters),iColours,true);
227
228     // ……… enter eventLoop
229
230     eventLoop();
231   }
232
233   // ########################################################################## doInitManagers
234
235   void  doInitManagers(void)
236   {
237     MaxApplZone();
238     MoreMasters();
239
240     InitGraf(&qd.thePort);
241     InitFonts();
242     InitWindows();
243     InitMenus();
244     TEInit();
245     InitDialogs(NULL);
246
247     InitCursor();
248     FlushEvents(everyEvent,0);
249   }
250
251   // ##################################################################### doOpenFloatingWindows
252
253   void  doOpenFloatingWindows(void)
254   {
255     SInt16          resourceOffset = 0;
256     PicHandle       pictureHdl;
257     ActivateProcPtr procPtr;
258
259     if(gColorQuickDrawPresent == false || gColourDisplay == false)
260       resourceOffset = 4;
261
262     procPtr = (ActivateProcPtr) &toolsActivateHandler;
263     gToolsWindowPtr = newGetNewWindow(rToolsWindow,(WindowPtr) -1,procPtr,kFloatingKind);
264     if(gToolsWindowPtr == NULL)
265     {
266       SysBeep(10);
267       ExitToShell();
268     }
269
270     pictureHdl = GetPicture(rToolsPict + resourceOffset);
271     SetWindowPic(gToolsWindowPtr,pictureHdl);
272
273     procPtr = (ActivateProcPtr) &coloursActivateHandler;
274     gColoursWindowPtr = newGetNewWindow(rColoursWindow,(WindowPtr) -1,procPtr,kFloatingKind);
275     if(gColoursWindowPtr == NULL)
276     {
277       SysBeep(10);
278       ExitToShell();
279     }
280
281     pictureHdl = GetPicture(rColoursPict + resourceOffset);
282     SetWindowPic(gColoursWindowPtr,pictureHdl);
283
```

```
  HiliteWindow(gToolsWindowPtr,true);
}

// ############################################################################## eventLoop

void  eventLoop(void)
{
  EventRecord eventRec;
  DialogPtr   theDialogPtr;
  SInt16      itemHit;

  gDone = false;

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&eventRec,MAXLONG,NULL))
    {
      if(!IsDialogEvent(&eventRec))
        doEvents(&eventRec);
      else
      {
        DialogSelect(&eventRec,&theDialogPtr,&itemHit);
        if((itemHit == ok) || (itemHit == cancel))
          doDisposeFindDialog();
      }
    }
  }
}

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

void  doEvents(EventRecord *eventRecPtr)
{
  SInt8 charCode;

  switch(eventRecPtr->what)
  {
    case mouseDown:
      doMouseDown(eventRecPtr);
      break;

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

    case updateEvt:
      doUpdate(eventRecPtr);
      break;

    case osEvt:
      doOSEvent(eventRecPtr);
      HiliteMenu(0);
      break;
  }
}

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

void  doMouseDown(EventRecord *eventRecPtr)
{
  SInt16          partCode;
  WindowPtr       windowPtr;
  WindowPtr       frontWindowPtr;
  docRecordHandle docRecordHdl;
  SInt16          windowType;
  UInt32          newSize;
  Rect            growRect;

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

  switch(partCode)
```

```
  {
    case inMenuBar:
      doAdjustMenus();
      doMenuChoice(MenuSelect(eventRecPtr->where));
      break;

    case inSysWindow:
      SystemClick(eventRecPtr,windowPtr);
      break;

    case inContent:
      frontWindowPtr = FrontWindow();
      if(((WindowPeek) frontWindowPtr)->windowKind != dialogKind)
        doProvingBeeps(windowPtr);
      if((isWindowModal(frontWindowPtr)) && (windowPtr != frontWindowPtr))
        SysBeep(10);
      else
        if(windowPtr != frontWindowPtr)
          newSelectWindow(windowPtr);
      break;

    case inDrag:
      frontWindowPtr = FrontWindow();
      if((isWindowModal(frontWindowPtr)) && (windowPtr != frontWindowPtr))
        SysBeep(10);
      else
        newDragWindow(windowPtr,eventRecPtr->where,&qd.screenBits.bounds);
      break;

    case inGoAway:
      if(TrackGoAway(windowPtr,eventRecPtr->where))
      {
        docRecordHdl = (docRecordHandle) GetWRefCon(windowPtr);
        windowType = (*docRecordHdl)->windowType;
        if(windowType == kFloatingKind)
        {
          newHideWindow(windowPtr);
          doAdjustMenus();
        }
        else if(windowType == kDocumentKind)
          doCloseDocWindow();
      }
      break;

    case inGrow:
      growRect = qd.screenBits.bounds;
      growRect.top = 145;
      growRect.left = 335;
      newSize = GrowWindow(windowPtr,eventRecPtr->where,&growRect);
      if(newSize != 0)
      {
        invalidateScrollBarArea(windowPtr);
        SizeWindow(windowPtr,LoWord(newSize),HiWord(newSize),true);
        invalidateScrollBarArea(windowPtr);
      }
      break;

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

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

void  doUpdate(EventRecord *eventRecPtr)
{
  WindowPtr windowPtr;

  windowPtr = (WindowPtr)eventRecPtr->message;
```

```
438     SetPort(windowPtr);
439
440     BeginUpdate(windowPtr);
441     EraseRect(&windowPtr->portRect);
442     doDrawDocWindowContent();
443     DrawGrowIcon(windowPtr);
444     EndUpdate(windowPtr);
445   }
446
447   // ############################################################################## doOSEvent
448
449   void  doOSEvent(EventRecord *eventRecPtr)
450   {
451     switch((eventRecPtr->message >>24) & 0x000000FF)
452     {
453       case suspendResumeMessage:
454         if(eventRecPtr->message & resumeFlag)
455         {
456           gInBackground = false;
457           handleResumeEvent();
458           SetCursor(&qd.arrow);
459         }
460         else
461         {
462           gInBackground = true;
463           handleSuspendEvent();
464         }
465         break;
466
467       case mouseMovedMessage:
468         break;
469     }
470   }
471
472   // ############################################################################ doAdjustMenus
473
474   void  doAdjustMenus(void)
475   {
476     MenuHandle  appleMenuHdl, fileMenuHdl, floatMenuHdl;
477
478     appleMenuHdl = GetMenuHandle(mApple);
479     fileMenuHdl = GetMenuHandle(mFile);
480     floatMenuHdl = GetMenuHandle(mFloaters);
481
482     if(isWindowModal(FrontWindow()))
483     {
484       DisableItem(appleMenuHdl,1);
485       DisableItem(fileMenuHdl,0);
486       DisableItem(floatMenuHdl,0);
487     }
488     else
489     {
490       EnableItem(appleMenuHdl,1);
491       EnableItem(fileMenuHdl,0);
492       EnableItem(floatMenuHdl,0);
493
494       if(gNumberDocWindowsOpen)
495         EnableItem(fileMenuHdl,iClose);
496       else
497         DisableItem(fileMenuHdl,iClose);
498     }
499
500     if(getWindowVisible(gToolsWindowPtr))
501       CheckItem(floatMenuHdl,iTools,true);
502     else
503       CheckItem(floatMenuHdl,iTools,false);
504
505     if(getWindowVisible(gColoursWindowPtr))
506       CheckItem(floatMenuHdl,iColours,true);
507     else
508       CheckItem(floatMenuHdl,iColours,false);
509
510     DrawMenuBar();
511   }
512
513   // ############################################################################## doMenuChoice
514
```

```
515 void  doMenuChoice(SInt32 menuChoice)
516 {
517   SInt16    menuID, menuItem;
518   Str255  itemName;
519   SInt16    daDriverRefNum;
520
521   menuID = HiWord(menuChoice);
522   menuItem = LoWord(menuChoice);
523
524   if(menuID == 0)
525     return;
526
527   switch(menuID)
528   {
529     case mApple:
530       if(menuItem == iAbout)
531       {
532         deactivateFloatsAndFirstDocWin();
533         NoteAlert(rAboutAlert,NULL);
534         activateFloatsAndFirstDocWin();
535       }
536       else
537       {
538         GetMenuItemText(GetMenuHandle(mApple),menuItem,itemName);
539         daDriverRefNum = OpenDeskAcc(itemName);
540       }
541       break;
542
543     case mFile:
544       doFileMenu(menuItem);
545       break;
546
547     case mFloaters:
548       doFloatersMenu(menuItem);
549       break;
550   }
551
552   HiliteMenu(0);
553 }
554
555 // ############################################################################## doFileMenu
556
557 void  doFileMenu(SInt16 menuItem)
558 {
559   switch(menuItem)
560   {
561     case iNew:
562       doOpenDocWindow();
563       break;
564
565     case iClose:
566       doCloseDocWindow();
567       break;
568
569     case iFind:
570       doOpenFindDialog();
571       break;
572
573     case iQuit:
574       gDone = true;
575       break;
576   }
577 }
578
579 // ############################################################################ doFloatersMenu
580
581 void  doFloatersMenu(SInt16 menuItem)
582 {
583   Boolean   floaterVisible;
584   WindowPtr floaterPicked;
585
586   if(menuItem == iTools)
587     floaterPicked = gToolsWindowPtr;
588   else if(menuItem == iColours)
589     floaterPicked = gColoursWindowPtr;
590
591   floaterVisible = getWindowVisible(floaterPicked);
```

```
592
593   if(floaterVisible == false)
594     newShowWindow(floaterPicked);
595   else
596     newHideWindow(floaterPicked);
597
598   doAdjustMenus();
599 }
600
601 // ########################################################################## doOpenDocWindow
602
603 void  doOpenDocWindow(void)
604 {
605   ActivateProcPtr procPtr;
606   WindowPtr       windowPtr;
607
608   if(gNumberDocWindowsOpen > 3)
609     SysBeep(10);
610   else
611   {
612     procPtr = (ActivateProcPtr) &docActivateHandler;
613     windowPtr = newGetNewWindow(rDocWindow,(WindowPtr) -1,procPtr,kDocumentKind);
614     newShowWindow(windowPtr);
615     gNumberDocWindowsOpen ++;
616   }
617 }
618
619 // ########################################################################## doCloseDocWindow
620
621 void  doCloseDocWindow(void)
622 {
623   WindowPtr   windowPtr;
624
625   windowPtr = findFrontNonFloatingWindow();
626
627   if(windowPtr != NULL)
628     newDisposeWindow(windowPtr);
629
630   gNumberDocWindowsOpen --;
631 }
632
633 // ############################################################### doDrawDocWindowContent
634
635 void  doDrawDocWindowContent(void)
636 {
637   Str255  theString;
638   SInt16  a;
639
640   for(a=1;a<4;a++)
641   {
642     GetIndString(theString,128,a);
643     MoveTo(20,a*30);
644     DrawString(theString);
645   }
646 }
647
648 // ########################################################################## doProvingBeeps
649
650 void  doProvingBeeps(WindowPtr windowPtr)
651 {
652   Handle  soundHdl;
653   SInt16  repeat, a;
654
655   soundHdl = GetResource('snd ',rSound);
656
657   if(windowPtr == gToolsWindowPtr)
658     repeat = 2;
659   else if(windowPtr == gColoursWindowPtr)
660     repeat = 3;
661   else
662     repeat = 1;
663
664   for(a=0;a<repeat;a++)
665     SndPlay(NULL,(SndListHandle) soundHdl,false);
666
667   ReleaseResource(soundHdl);
668 }
```

```
669
670 // ########################################################################## doOpenFindDialog
671
672 void  doOpenFindDialog(void)
673 {
674   DialogPtr findDialogPtr;
675
676   deactivateFloatsAndFirstDocWin();
677   findDialogPtr = GetNewDialog(rFindDialog,NULL,(WindowPtr) - 1);
678
679   doAdjustMenus();
680 }
681
682 // ########################################################################## doDisposeFindDialog
683
684 void  doDisposeFindDialog(void)
685 {
686   DialogPtr findDialogPtr;
687
688   findDialogPtr = FrontWindow();
689   DisposeDialog(findDialogPtr);
690
691   activateFloatsAndFirstDocWin();
692
693   doAdjustMenus();
694 }
695
696 // ########################################################################## docActivateHandler
697
698 pascal void  docActivateHandler(WindowPtr windowPtr,Boolean activate)
699 {
700   GrafPtr oldPort;
701   SInt32  finalTicks;
702   Rect    theRect;
703
704   GetPort(&oldPort);
705   SetPort(windowPtr);
706
707   DrawGrowIcon(windowPtr);
708
709   MoveTo(20,120);
710   if(activate)
711     DrawString("\pActivating");
712   else
713     DrawString("\pDeactivating");
714
715   Delay(30,&finalTicks);
716
717   SetRect(&theRect,20,105,100,125);
718   EraseRect(&theRect);
719
720   SetPort(oldPort);
721 }
722
723 // ########################################################################## toolsActivateHandler
724
725 pascal void  toolsActivateHandler(WindowPtr windowPtr,Boolean activate)
726 {
727   GrafPtr   oldPort;
728   SInt16    resourceOffset = 0;
729   PicHandle pictureHdl;
730   SInt16    whichPicture;
731   Rect      theRect;
732
733   GetPort(&oldPort);
734   SetPort(windowPtr);
735
736   if(activate == true)
737     whichPicture = rToolsPict;
738   else
739     whichPicture = rToolsPictDim;
740
741   if(gColorQuickDrawPresent == false || gColourDisplay == false)
742     resourceOffset = 4;
743
744   pictureHdl = GetPicture(whichPicture + resourceOffset);
745   theRect = (*pictureHdl)->picFrame;
```

```
  OffsetRect(&theRect,- theRect.left, - theRect.top);
  if(gColorQuickDrawPresent == false || gColourDisplay == false)
    EraseRect(&windowPtr->portRect);
  DrawPicture(pictureHdl,&theRect);
  SetWindowPic(windowPtr,pictureHdl);

  SetPort(oldPort);
}

// ############################################################## coloursActivateHandler

pascal void  coloursActivateHandler(WindowPtr windowPtr,Boolean activate)
{
  GrafPtr   oldPort;
  SInt16    resourceOffset = 0;
  PicHandle pictureHdl;
  SInt16    whichPicture;
  Rect      theRect;

  GetPort(&oldPort);
  SetPort(windowPtr);

  if(activate == true)
    whichPicture = rColoursPict;
  else
    whichPicture = rColoursPictDim;

  if(gColorQuickDrawPresent == false || gColourDisplay == false)
    resourceOffset = 4;

  pictureHdl = GetPicture(whichPicture + resourceOffset);
  theRect = (*pictureHdl)->picFrame;
  OffsetRect(&theRect,- theRect.left, - theRect.top);
  if(gColorQuickDrawPresent == false || gColourDisplay == false)
    EraseRect(&windowPtr->portRect);
  DrawPicture(pictureHdl,&theRect);
  SetWindowPic(windowPtr,pictureHdl);

  SetPort(oldPort);
}

// ############################################################## invalidateScrollBarArea

void  invalidateScrollBarArea(WindowPtr windowPtr)
{
  Rect  tempRect;

  SetPort(windowPtr);

  tempRect = windowPtr->portRect;
  tempRect.left = tempRect.right - 15;
  InvalRect(&tempRect);

  tempRect = windowPtr->portRect;
  tempRect.top = tempRect.bottom - 15;
  InvalRect(&tempRect);
}

// ############################################################################################

// ############################################################################################
// FloatRoutines.c                                        Routines to support floating windows
// ############################################################################################

// ……… includes

#include "Floaters.h"

// ##################################################################### newGetNewWindow

WindowPtr  newGetNewWindow(SInt16 windResourceID,WindowPtr behind,
                           ActivateProcPtr activateHandler,SInt16 windowType)
{
  docRecordHandle docRecordHdl;
  OSErr           osErr;
  SInt32          response;
  WindowPtr       newWindowPtr;
```

```
  WindowPtr       lastFloatingWindowPtr;

  docRecordHdl = (docRecordHandle) NewHandle(sizeof(docRecord));

  osErr = Gestalt(gestaltQuickdrawVersion,&response);
  if(response < gestalt32BitQD)
    newWindowPtr = GetNewWindow(windResourceID,NULL,(WindowPtr) behind);
  else
    newWindowPtr = GetNewCWindow(windResourceID,NULL,(WindowPtr) behind);

  if(newWindowPtr != NULL)
  {
    SetWRefCon(newWindowPtr,(SInt32) docRecordHdl);
    setActivateHandler(newWindowPtr,activateHandler);

    if(windowType == kFloatingKind)
    {
      setWindowKind(newWindowPtr,kFloatingKind);
      HiliteWindow(newWindowPtr,true);
    }
    else
    {
      setWindowKind(newWindowPtr,kDocumentKind);
      if(behind == (WindowPtr) -1)
      {
        lastFloatingWindowPtr = findLastFloatingWindow();

        if(lastFloatingWindowPtr != NULL)
          SendBehind(newWindowPtr,lastFloatingWindowPtr);
        else
          BringToFront(newWindowPtr);
      }
    }
  }
  else
    DisposeHandle((Handle) docRecordHdl);

  return newWindowPtr;
}

// ########################################################################## newDisposeWindow

void  newDisposeWindow(WindowPtr windowPtr)
{
  if(getWindowVisible(windowPtr))
    newHideWindow(windowPtr);
  CloseWindow(windowPtr);
  DisposeHandle((Handle) GetWRefCon(windowPtr));
  DisposePtr((Ptr) windowPtr);
}

// ########################################################################## newSelectWindow

void  newSelectWindow(WindowPtr windowToSelectPtr)
{
  Boolean   isFloatingWindow;
  WindowPtr currentFrontWindowPtr;
  WindowPtr lastFloatingWindowPtr;

  if(getWindowKind(windowToSelectPtr) == kFloatingKind)
  {
    isFloatingWindow = true;
    currentFrontWindowPtr = FrontWindow();
  }
  else
  {
    isFloatingWindow = false;
    currentFrontWindowPtr = findFrontNonFloatingWindow();
    lastFloatingWindowPtr = findLastFloatingWindow();
  }

  if(currentFrontWindowPtr != windowToSelectPtr)
  {
    if(isFloatingWindow)
      BringToFront(windowToSelectPtr);
    else
    {
```

```
      if(lastFloatingWindowPtr == NULL)
        SelectWindow(windowToSelectPtr);
      else
      {
        deactivateWindow(currentFrontWindowPtr);
        SendBehind(windowToSelectPtr,lastFloatingWindowPtr);
        activateWindow(windowToSelectPtr);
      }
    }
  }
}

// ########################################################################## newHideWindow

void  newHideWindow(WindowPtr windowToHidePtr)
{
  WindowPtr frontFloaterPtr;
  WindowPtr frontNonFloaterPtr;
  WindowPtr windowBehindPtr;
  WindowPtr lastFloaterPtr;

  if(getWindowVisible(windowToHidePtr) == false)
    return;

  frontFloaterPtr = FrontWindow();
  if(getWindowKind(frontFloaterPtr) != kFloatingKind)
    frontFloaterPtr = NULL;

  frontNonFloaterPtr = findFrontNonFloatingWindow();

  ShowHide(windowToHidePtr,false);

  if(windowToHidePtr == frontFloaterPtr)
  {
    windowBehindPtr = getNextWindow(windowToHidePtr);

    if((windowBehindPtr != NULL) && (getWindowKind(windowBehindPtr) == kFloatingKind))
    {
      setNextWindow(windowToHidePtr,getNextWindow(windowBehindPtr));
      setNextWindow(windowBehindPtr,windowToHidePtr);
      setWindowList(windowBehindPtr);
    }
  }
  else
  {
    if(windowToHidePtr == frontNonFloaterPtr)
    {
      windowBehindPtr = getNextWindow(windowToHidePtr);

      if(windowBehindPtr != NULL)
      {
        setNextWindow(windowToHidePtr,getNextWindow(windowBehindPtr));
        setNextWindow(windowBehindPtr,windowToHidePtr);

        lastFloaterPtr = findLastFloatingWindow();
        if(lastFloaterPtr != NULL)
          setNextWindow(lastFloaterPtr,windowBehindPtr);
        else
          setWindowList(windowBehindPtr);

        activateWindow(windowBehindPtr);
      }
    }
  }
}

// ########################################################################## newShowWindow

void  newShowWindow(WindowPtr windowToShowPtr)
{
  SInt16          windowType;
  WindowPtr       windowBehindPtr;
  WindowPtr       frontNonFloatingWindowPtr;
  Boolean         windowIsInFront = false;
  ActivateProcPtr activateHandler;

  if(getWindowVisible(windowToShowPtr) != false)
```

```
977       return;
978
979     windowType = getWindowKind(windowToShowPtr);
980
981     if(windowType != kFloatingKind)
982     {
983       windowBehindPtr = getNextWindow(windowToShowPtr);
984       if(windowBehindPtr == findFrontNonFloatingWindow())
985       {
986         if(windowBehindPtr != NULL)
987           deactivateWindow(windowBehindPtr);
988
989         setWindowHilite(windowToShowPtr,true);
990         windowIsInFront = true;
991       }
992     }
993     else
994     {
995       frontNonFloatingWindowPtr = findFrontNonFloatingWindow();
996
997       if((frontNonFloatingWindowPtr != NULL) &&
998          (frontNonFloatingWindowPtr == FrontWindow()) &&
999          (isWindowModal(frontNonFloatingWindowPtr)))
1000      {
1001        setWindowHilite(windowToShowPtr,false);
1002      }
1003      else
1004      {
1005        setWindowHilite(windowToShowPtr,true);
1006        windowIsInFront = true;
1007      }
1008    }
1009
1010    ShowHide(windowToShowPtr,true);
1011
1012    if(windowIsInFront)
1013    {
1014      activateHandler = getActivateHandler(windowToShowPtr);
1015      (*activateHandler) (windowToShowPtr,true);
1016    }
1017  }
1018
1019  // ########################################################################## newDragWindow
1020
1021  void  newDragWindow(WindowPtr windowPtr,Point startPoint,const Rect *dragBounds)
1022  {
1023    SInt16    topLimit;
1024    Rect      slopRect;
1025    GrafPtr   oldPort;
1026    GrafPtr   windowManagerPort;
1027    KeyMap    keyMap;
1028    Boolean   commandKeyDown = false;
1029    RgnHandle dragRegion;
1030    SInt32    dragResult;
1031    SInt16    horizOffset;
1032    SInt16    vertOffset;
1033    RgnHandle windowContentRegion;
1034    SInt16    newHorizWindowPosition;
1035    SInt16    newVertWindowPosition;
1036
1037    if(WaitMouseUp())
1038    {
1039      topLimit = GetMBarHeight() + 4;
1040      slopRect = *dragBounds;
1041
1042      if(slopRect.top < topLimit)
1043        slopRect.top = topLimit;
1044
1045      GetPort(&oldPort);
1046      GetWMgrPort(&windowManagerPort);
1047      SetPort(windowManagerPort);
1048
1049      SetClip(GetGrayRgn());
1050
1051      GetKeys(keyMap);
1052      if(keyMap[1] & 0x8000)
1053        commandKeyDown = true;
```

```
1054
1055      if((commandKeyDown == true) || (getWindowKind(windowPtr) != kFloatingKind))
1056      {
1057        if (commandKeyDown == false)
1058          ClipAbove((WindowRef) ((WindowPeek) findFrontNonFloatingWindow()));
1059        else
1060          ClipAbove((WindowRef) ((WindowPeek) windowPtr));
1061      }
1062
1063      dragRegion = NewRgn();
1064      CopyRgn(getStructureRegion(windowPtr), dragRegion);
1065
1066      dragResult = DragGrayRgn(dragRegion,startPoint,&slopRect,&slopRect,noConstraint,nil);
1067
1068      SetPort(oldPort);
1069
1070      if(dragResult != 0)
1071      {
1072        horizOffset = dragResult & 0xFFFF;
1073        vertOffset = dragResult >> 16;
1074
1075        if(vertOffset != -32768)
1076        {
1077          windowContentRegion = getContentRegion(windowPtr);
1078
1079          newHorizWindowPosition = (**windowContentRegion).rgnBBox.left +  horizOffset;
1080          newVertWindowPosition = (**windowContentRegion).rgnBBox.top + vertOffset;
1081
1082          MoveWindow(windowPtr,newHorizWindowPosition,newVertWindowPosition,false);
1083        }
1084      }
1085
1086      if(commandKeyDown == false)
1087        newSelectWindow(windowPtr);
1088
1089      DisposeRgn(dragRegion);
1090    }
1091  }
1092
1093  // ########################################################################## handleSuspendEvent
1094
1095  void  handleSuspendEvent(void)
1096  {
1097    WindowPtr currentWindowPtr;
1098    Boolean   windowIsVisible;
1099
1100    currentWindowPtr = getWindowList();
1101
1102    if(getWindowKind(currentWindowPtr) != kFloatingKind)
1103      return;
1104
1105    do
1106    {
1107      windowIsVisible = getWindowVisible(currentWindowPtr);
1108      setWasVisible(currentWindowPtr,windowIsVisible);
1109      if(windowIsVisible)
1110        ShowHide(currentWindowPtr,false);
1111      currentWindowPtr = getNextWindow(currentWindowPtr);
1112    } while((currentWindowPtr != NULL) &&
1113            (getWindowKind(currentWindowPtr) == kFloatingKind));
1114
1115    currentWindowPtr = findFrontNonFloatingWindow();
1116    if(currentWindowPtr != NULL)
1117    {
1118      DrawGrowIcon(currentWindowPtr);
1119      deactivateWindow(currentWindowPtr);
1120    }
1121  }
1122
1123  // ########################################################################## handleResumeEvent
1124
1125  void  handleResumeEvent(void)
1126  {
1127    WindowPtr currentWindowPtr;
1128    Boolean   windowWasVisible;
1129
1130    currentWindowPtr = getWindowList();
```

```
1131
1132    if(getWindowKind(currentWindowPtr) != kFloatingKind)
1133      return;
1134
1135    do
1136    {
1137      windowWasVisible = getWasVisible(currentWindowPtr);
1138      if(windowWasVisible)
1139      {
1140        ShowHide(currentWindowPtr,true);
1141        activateWindow(currentWindowPtr);
1142      }
1143      currentWindowPtr = getNextWindow(currentWindowPtr);
1144    } while((currentWindowPtr != NULL) &&
1145            (getWindowKind(currentWindowPtr) == kFloatingKind));
1146
1147    currentWindowPtr = findFrontNonFloatingWindow();
1148    if(currentWindowPtr != NULL)
1149    {
1150      DrawGrowIcon(currentWindowPtr);
1151      activateWindow(currentWindowPtr);
1152    }
1153  }
1154
1155  // ######################################################## deactivateFloatsAndFirstDocWin
1156
1157  void  deactivateFloatsAndFirstDocWin(void)
1158  {
1159    WindowPtr firstWindowPtr;
1160    WindowPtr secondDocumentWindowPtr;
1161    WindowPtr currentWindowPtr;
1162
1163    firstWindowPtr = FrontWindow();
1164    secondDocumentWindowPtr = findFrontNonFloatingWindow();
1165    if(secondDocumentWindowPtr != NULL)
1166      secondDocumentWindowPtr = getNextWindow(secondDocumentWindowPtr);
1167
1168    currentWindowPtr = firstWindowPtr;
1169    while(currentWindowPtr != secondDocumentWindowPtr)
1170    {
1171      if(getWindowVisible(currentWindowPtr))
1172        deactivateWindow(currentWindowPtr);
1173      currentWindowPtr = getNextWindow(currentWindowPtr);
1174    }
1175  }
1176
1177  // ######################################################## activateFloatsAndFirstDocWin
1178
1179  void  activateFloatsAndFirstDocWin(void)
1180  {
1181    OSErr               getFrontProcessResult;
1182    OSErr               getCurrentProcessResult;
1183    ProcessSerialNumber frontPSN;
1184    ProcessSerialNumber currentPSN;
1185    OSErr               sameProcessResult;
1186    Boolean             isSameProcess;
1187    WindowPtr           firstWindowPtr;
1188    WindowPtr           secondDocumentWindowPtr;
1189    WindowPtr           currentWindowPtr;
1190
1191    getFrontProcessResult = GetFrontProcess(&frontPSN);
1192    getCurrentProcessResult = GetCurrentProcess(&currentPSN);
1193
1194    if((getFrontProcessResult == noErr) && (getCurrentProcessResult == noErr))
1195      sameProcessResult = SameProcess(&frontPSN,&currentPSN,&isSameProcess);
1196
1197    if((sameProcessResult == noErr) && (isSameProcess == false))
1198      handleSuspendEvent();
1199    else
1200    {
1201      firstWindowPtr = FrontWindow();
1202      secondDocumentWindowPtr = findFrontNonFloatingWindow();
1203      if(secondDocumentWindowPtr != NULL)
1204        secondDocumentWindowPtr = getNextWindow(secondDocumentWindowPtr);
1205      currentWindowPtr = firstWindowPtr;
1206
1207      while(currentWindowPtr != secondDocumentWindowPtr)
```

```
    {
      if(getWindowVisible(currentWindowPtr))
        activateWindow(currentWindowPtr);
      currentWindowPtr = getNextWindow(currentWindowPtr);
    }
  }
}

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

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

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

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

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

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

  HiliteWindow(windowPtr,activate);
  activateHandler = getActivateHandler(windowPtr);
  (*activateHandler) (windowPtr,activate);
}

// ############################################################## findFrontNonFloatingWindow

WindowPtr  findFrontNonFloatingWindow(void)
{
  WindowPtr frontWindowPtr;

  frontWindowPtr = FrontWindow();

  while((frontWindowPtr != NULL) && (getWindowKind(frontWindowPtr) == kFloatingKind))
  {
    do
    {
      frontWindowPtr = getNextWindow(frontWindowPtr);
    } while((frontWindowPtr != NULL) && (getWindowVisible(frontWindowPtr) == false));
  }

  return(frontWindowPtr);
}

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

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

  windowPtr = getWindowList();
  lastFloatingWindowPtr = NULL;

  while(windowPtr != NULL)
  {
    if(getWindowKind(windowPtr) == kFloatingKind)
      lastFloatingWindowPtr = windowPtr;
    windowPtr = getNextWindow(windowPtr);
  }

  return lastFloatingWindowPtr;
}

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

Boolean  isWindowModal(WindowPtr windowPtr)
{
  SInt16  variant;
```

```
1285
1286    variant = GetWVariant(windowPtr);
1287
1288    if((((WindowPeek) windowPtr)->windowKind == dialogKind) &&
1289       ((variant == dBoxProc) || (variant == movableDBoxProc)))
1290      return true;
1291    else
1292      return false;
1293  }
1294
1295  // ########################################################################## getWindowList
1296
1297  WindowPtr  getWindowList(void)
1298  {
1299    return(LMGetWindowList());
1300  }
1301
1302  // ########################################################################## setWindowList
1303
1304  void  setWindowList(WindowPtr windowPtr)
1305  {
1306    LMSetWindowList(windowPtr);
1307  }
1308
1309  // ########################################################################## getNextWindow
1310
1311  WindowPtr  getNextWindow(WindowPtr windowPtr)
1312  {
1313    return((WindowPtr) ((WindowPeek) windowPtr)->nextWindow);
1314  }
1315
1316  // ########################################################################## setNextWindow
1317
1318  void  setNextWindow(WindowPtr windowPtr,WindowPtr nextWindowPtr)
1319  {
1320    ((WindowPeek) windowPtr)->nextWindow = (WindowPeek) nextWindowPtr;
1321  }
1322
1323  // ########################################################################## getWasVisible
1324
1325  Boolean  getWasVisible(WindowPtr windowPtr)
1326  {
1327    docRecordHandle docRecordHdl;
1328
1329    docRecordHdl = (docRecordHandle) GetWRefCon(windowPtr);
1330    return((*docRecordHdl)->wasVisible);
1331  }
1332
1333  // ########################################################################## setWasVisible
1334
1335  void  setWasVisible(WindowPtr windowPtr,Boolean wasVisible)
1336  {
1337    docRecordHandle docRecordHdl;
1338
1339    docRecordHdl = (docRecordHandle) GetWRefCon(windowPtr);
1340    (*docRecordHdl)->wasVisible = wasVisible;
1341  }
1342
1343  // ########################################################################## getWindowKind
1344
1345  SInt16  getWindowKind(WindowPtr windowPtr)
1346  {
1347    docRecordHandle docRecordHdl;
1348
1349    if(((WindowPeek) windowPtr)->windowKind != dialogKind)
1350    {
1351      docRecordHdl = (docRecordHandle) GetWRefCon(windowPtr);
1352      return((*docRecordHdl)->windowType);
1353    }
1354    else
1355      return(0);
1356  }
1357
1358  // ########################################################################## setWindowKind
1359
1360  void  setWindowKind(WindowPtr windowPtr,SInt16 windowKind)
1361  {
```

```
1362    docRecordHandle docRecordHdl;
1363
1364    docRecordHdl = (docRecordHandle) GetWRefCon(windowPtr);
1365    (*docRecordHdl)->windowType = windowKind;
1366  }
1367
1368  // ########################################################################## getWindowVisible
1369
1370  Boolean  getWindowVisible(WindowPtr windowPtr)
1371  {
1372    return(((WindowPeek) windowPtr)->visible);
1373  }
1374
1375  // ########################################################################## getStructureRegion
1376
1377  RgnHandle  getStructureRegion(WindowPtr windowPtr)
1378  {
1379    return(((WindowPeek) windowPtr)->strucRgn);
1380  }
1381
1382  // ########################################################################## getContentRegion
1383
1384  RgnHandle  getContentRegion(WindowPtr windowPtr)
1385  {
1386    return(((WindowPeek) windowPtr)->contRgn);
1387  }
1388
1389  // ########################################################################## setWindowHilite
1390
1391  void  setWindowHilite(WindowPtr windowPtr,Boolean windowHilite)
1392  {
1393    ((WindowPeek) windowPtr)->hilited = windowHilite;
1394  }
1395
1396  // ########################################################################## getActivateHandler
1397
1398  ActivateProcPtr  getActivateHandler(WindowPtr windowPtr)
1399  {
1400    docRecordHandle docRecordHdl;
1401
1402    docRecordHdl = (docRecordHandle) GetWRefCon(windowPtr);
1403    return((*docRecordHdl)->activateHandler);
1404  }
1405
1406  // ########################################################################## setActivateHandler
1407
1408  void  setActivateHandler(WindowPtr windowPtr,ActivateProcPtr activateHandler)
1409  {
1410    docRecordHandle docRecordHdl;
1411
1412    docRecordHdl = (docRecordHandle) GetWRefCon(windowPtr);
1413    (*docRecordHdl)->activateHandler = activateHandler;
1414  }
1415
1416  // ############################################################################################
1417
1418  // ############################################################################################
1419  // WDEF.c                          Custom Window Definition Function for Floating Windows
1420  // ############################################################################################
1421  //
1422  // This WDEF creates a utility window whose appearance conforms to that specified in the
1423  // document titled Apple Grayscale Appearance for System 7.5 and published by Apple
1424  // Computer, Inc.  On black-and-white displays, the WDEF is drawn in black-and-white with
1425  // an appearance similar to the black-and-white floating windows found in many commercial
1426  // applications.  The WDEF supports only one variation code.  It provides for a close box
1427  // but not for a zoom box or window title.
1428  //
1429  // The WDEF utilises three 'cicn' resources (unpurgeable), one for the close box in the
1430  // normal state, one for the close box in the pressed state, and one to paint the
1431  // checkered pattern in the title bar.
1432  //
1433  // ############################################################################################
1434
1435  // …… includes
1436
1437  #include <A4Stuff.h>
1438  #include <Gestalt.h>
```

```
#include <ToolUtils.h>
#include <LowMem.h>

// ……… defines

#define rCloseEnabledIcon 128
#define rClosePressedIcon 129
#define rCheckPatternIcon 130

// ……… global variables

Boolean     gColorQuickDrawPresent = false;
Boolean     gColourDisplay         = false;
Pattern     gBlackPattern;
CIconHandle gCloseEnabledHdl;
CIconHandle gClosePressedHdl;
CIconHandle gCheckPatternHdl;
RGBColor    gWhite  = { 0xFFFF,0xFFFF,0xFFFF };
RGBColor    gGray1  = { 0xEEEE,0xEEEE,0xEEEE };
RGBColor    gGray2  = { 0xDDDD,0xDDDD,0xDDDD };
RGBColor    gGray3  = { 0xCCCC,0xCCCC,0xCCCC };
RGBColor    gGray4  = { 0xBBBB,0xBBBB,0xBBBB };
RGBColor    gGray6  = { 0x9999,0x9999,0x9999 };
RGBColor    gGray7  = { 0x8888,0x8888,0x8888 };
RGBColor    gGray8  = { 0x7777,0x7777,0x7777 };
RGBColor    gGray10 = { 0x5555,0x5555,0x5555 };
RGBColor    gBlack  = { 0x0000,0x0000,0x0000 };
Boolean     gToggle;

// ……… function prototypes

void    doInitMessage         (void);
void    doDrawMessage         (WindowPeek,SInt32);
SInt32  doHitMessage          (WindowPeek,SInt32);
void    doCalcRgnsMessage     (WindowPeek);
void    drawWindowColour      (WindowPeek);
void    drawWindowMono        (WindowPeek);
void    toggleGoAway          (WindowPeek);
void    drawGoAwayBox         (WindowPeek);
void    drawGoAwayBoxPressed  (WindowPeek);
void    getGoAwayRect         (WindowPeek,Rect *);
void    getContentRect        (WindowPeek,Rect *);
void    getStructRect         (WindowPeek,Rect *);
void    syncPorts             (void);

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

pascal SInt32  main(SInt16 varCode,WindowPeek windowPeek,SInt16 message,SInt32 param)
{
  SInt32    result;
  GrafPtr   oldPort;
  PenState  oldPenState;

  EnterCodeResource();

  GetPenState(&oldPenState);
  GetPort(&oldPort);

  if(gColorQuickDrawPresent)
    syncPorts();

  result = 0;

  switch(message)
  {
    case wNew:
      doInitMessage();
      break;

    case wDraw:
      if(windowPeek->visible)
        doDrawMessage(windowPeek,param);
      break;

    case wHit:
      result = doHitMessage(windowPeek,param);
      break;
```

```
1516
1517      case wCalcRgns:
1518        doCalcRgnsMessage(windowPeek);
1519        break;
1520    }
1521
1522    SetPenState(&oldPenState);
1523    SetPort(oldPort);
1524
1525    ExitCodeResource();
1526
1527    return(result);
1528  }
1529
1530  // ############################################################################## doInitMessage
1531
1532  void  doInitMessage(void)
1533  {
1534    OSErr     osErr;
1535    SInt32    response;
1536    GDHandle  mainDeviceHdl;
1537    SInt16    bitsPerPixel, a;
1538
1539    osErr = Gestalt(gestaltQuickdrawVersion,&response);
1540    if(response >= gestalt8BitQD)
1541    {
1542      gColorQuickDrawPresent = true;
1543
1544      mainDeviceHdl = LMGetMainDevice();
1545      bitsPerPixel = (*(*mainDeviceHdl)->gdPMap)->pixelSize;
1546      if(bitsPerPixel > 1)
1547        gColourDisplay = true;
1548    }
1549
1550    for(a=0;a<8;a++)
1551      gBlackPattern.pat[a] = 0xFF;
1552
1553    gCloseEnabledHdl = GetCIcon(rCloseEnabledIcon);
1554    gClosePressedHdl = GetCIcon(rClosePressedIcon);
1555    gCheckPatternHdl = GetCIcon(rCheckPatternIcon);
1556
1557    gToggle = false;
1558  }
1559
1560  // ############################################################################## doDrawMessage
1561
1562  void  doDrawMessage(WindowPeek windowPeek,SInt32 param)
1563  {
1564    param &= 0x0000FFFF;
1565
1566    switch(param)
1567    {
1568      case wNoHit:
1569        if(gColorQuickDrawPresent && gColourDisplay)
1570          drawWindowColour(windowPeek);
1571        else
1572          drawWindowMono(windowPeek);
1573        break;
1574
1575      case wInGoAway:
1576        toggleGoAway(windowPeek);
1577        break;
1578
1579      default:
1580        return;
1581    }
1582  }
1583
1584  // ############################################################################## doHitMessage
1585
1586  SInt32  doHitMessage(WindowPeek windowPeek,SInt32 param)
1587  {
1588    Point where;
1589    Rect  goAwayRect;
1590
1591    where.v = HiWord(param);
1592    where.h = LoWord(param);
```

```
1593
1594    if(PtInRgn(where,windowPeek->contRgn))
1595      return(wInContent);
1596    else if(PtInRgn(where,windowPeek->strucRgn))
1597    {
1598      if(windowPeek->goAwayFlag)
1599      {
1600        getGoAwayRect(windowPeek,&goAwayRect);
1601        if(PtInRect(where,&goAwayRect))
1602          return(wInGoAway);
1603      }
1604
1605      return(wInDrag);
1606    }
1607
1608    return(wNoHit);
1609  }
1610
1611  // ########################################################################## doCalcRgnsMessage
1612
1613  void  doCalcRgnsMessage(WindowPeek windowPeek)
1614  {
1615    RgnHandle tempRgn;
1616    Rect      theRect;
1617
1618    tempRgn = NewRgn();
1619
1620    getContentRect(windowPeek,&theRect);
1621    RectRgn(windowPeek->contRgn,&theRect);
1622
1623    getStructRect(windowPeek,&theRect);
1624    RectRgn(windowPeek->strucRgn,&theRect);
1625    OffsetRect(&theRect,1,1);
1626    theRect.left += 1;
1627    theRect.top += 1;
1628    RectRgn(tempRgn,&theRect);
1629    UnionRgn(tempRgn,windowPeek->strucRgn,windowPeek->strucRgn);
1630
1631    DisposeRgn(tempRgn);
1632  }
1633
1634  // ########################################################################## drawWindowColour
1635
1636  void  drawWindowColour(WindowPeek windowPeek)
1637  {
1638    RGBColor  oldForeColour;
1639    RGBColor  oldBackColour;
1640    Rect      contentRect, structRect, theRect;
1641    SInt16    a, b;
1642
1643    GetForeColor(&oldForeColour);
1644    GetBackColor(&oldBackColour);
1645    PenSize(1,1);
1646    PenPat(&gBlackPattern);
1647    PenMode(patCopy);
1648
1649    if(windowPeek->hilited)
1650    {
1651      getContentRect(windowPeek,&contentRect);
1652
1653      RGBForeColor(&gBlack);
1654      InsetRect(&contentRect,-1,-1);
1655      FrameRect(&contentRect);
1656
1657      getStructRect(windowPeek,&structRect);
1658
1659      RGBForeColor(&gGray3);
1660      SetRect(&theRect,structRect.left + 2,structRect.top + 2,structRect.right - 2,
1661              structRect.top + 12);
1662      PaintRect(&theRect);
1663
1664      SetRect(&theRect,structRect.left + 14,structRect.top + 3,structRect.left + 22,
1665            structRect.top + 11);
1666      b = (((structRect.right - 4) - (structRect.left + 14)) / 9) + 1;
1667      for(a=0;a<b+1;a++)
1668      {
1669        PlotCIcon(&theRect,gCheckPatternHdl);
```

```
      OffsetRect(&theRect,9,0);
    }

    if(windowPeek->goAwayFlag)
      drawGoAwayBox(windowPeek);

    RGBForeColor(&gBlack);
    FrameRect(&structRect);
    MoveTo(structRect.left + 2,structRect.bottom);
    LineTo(structRect.right,structRect.bottom);
    LineTo(structRect.right,structRect.top + 2);

    RGBForeColor(&gWhite);
    MoveTo(structRect.left + 1,structRect.bottom - 3);
    LineTo(structRect.left + 1,structRect.top + 1);
    LineTo(structRect.right - 3,structRect.top + 1);
    MoveTo(structRect.left + 3,structRect.top + 11);
    LineTo(structRect.left + 11,structRect.top + 11);
    LineTo(structRect.left + 11,structRect.top + 3);

    RGBForeColor(&gGray1);
    MoveTo(structRect.left + 2,structRect.bottom - 3);
    LineTo(structRect.right - 3,structRect.bottom - 3);
    LineTo(structRect.right - 3,structRect.top + 13);

    RGBForeColor(&gGray3);
    MoveTo(structRect.right - 2,structRect.top + 1);
    LineTo(structRect.right - 2,structRect.top + 1);
    RGBForeColor(&gGray4);
    MoveTo(structRect.left + 1,structRect.bottom - 2);
    LineTo(structRect.left + 1,structRect.bottom - 2);

    RGBForeColor(&gGray6);
    MoveTo(structRect.left + 2,structRect.bottom - 2);
    LineTo(structRect.right - 2,structRect.bottom - 2);
    LineTo(structRect.right - 2,structRect.top + 2);
    MoveTo(structRect.right - 3,structRect.top + 12);
    LineTo(structRect.left + 2,structRect.top + 12);
    LineTo(structRect.left + 2,structRect.bottom - 4);

    RGBForeColor(&gGray7);
    MoveTo(structRect.left + 2,structRect.top + 10);
    LineTo(structRect.left + 2,structRect.top + 2);
    LineTo(structRect.left + 10,structRect.top + 2);
  }
  else
  {
    RGBForeColor(&gGray10);
    getContentRect(windowPeek,&contentRect);
    InsetRect(&contentRect,-1,-1);
    FrameRect(&contentRect);

    getStructRect(windowPeek,&structRect);
    FrameRect(&structRect);
    MoveTo(structRect.left + 2,structRect.bottom);
    LineTo(structRect.right,structRect.bottom);
    LineTo(structRect.right,structRect.top + 2);

    RGBForeColor(&gGray2);
    InsetRect(&structRect,1,1);
    PenSize(2,2);
    FrameRect(&structRect);
    structRect.bottom = structRect.top + 12;
    PaintRect(&structRect);
  }

  RGBForeColor(&oldForeColour);
  RGBBackColor(&oldBackColour);
}

// ########################################################################## drawWindowMono

void  drawWindowMono(WindowPeek windowPeek)
{
  Rect    contentRect, structRect, theRect;
  SInt16  a, b;
  UInt8   pattern;
```

```
  Pattern checkPattern;

  PenSize(1,1);
  PenPat(&gBlackPattern);
  PenMode(patCopy);

  if(windowPeek->goAwayFlag && windowPeek->hilited)
  {
    ForeColor(blackColor);
    BackColor(whiteColor);

    getContentRect(windowPeek,&contentRect);

    InsetRect(&contentRect,-1,-1);
    FrameRect(&contentRect);

    getStructRect(windowPeek,&structRect);

    FrameRect(&structRect);
    MoveTo(structRect.left + 2,structRect.bottom);
    LineTo(structRect.right,structRect.bottom);
    LineTo(structRect.right,structRect.top + 2);

    SetRect(&theRect,structRect.left + 1,structRect.top + 1, structRect.right - 1,
            structRect.top + 10);
    EraseRect(&theRect);

    for(a=0;a<8;a++)
      checkPattern.pat[a] = 0x00;

    if(structRect.left & 1)
      pattern = 0xAA;
    else
      pattern = 0x55;

    if(structRect.top & 1)
      b = 1;
    else
      b = 0;

    for(a=b;a<8;a+=2)
      checkPattern.pat[a] = pattern;

    PenPat(&checkPattern);
    SetRect(&theRect,structRect.left + 11,structRect.top + 2, structRect.right - 2,
            structRect.top + 9);
    PaintRect(&theRect);

    PenPat(&gBlackPattern);

    if(windowPeek->goAwayFlag)
      drawGoAwayBox(windowPeek);
  }
  else
  {
    SetRect(&theRect,structRect.left + 1,structRect.top + 1, structRect.right - 1,
            structRect.top + 13);
    EraseRect(&theRect);
  }
}

// ########################################################################## toggleGoAway

void  toggleGoAway(WindowPeek windowPeek)
{
  gToggle = !gToggle;

  if(gToggle)
    drawGoAwayBoxPressed(windowPeek);
  else
    drawGoAwayBox(windowPeek);
}

// ########################################################################## drawGoAwayBox

void  drawGoAwayBox(WindowPeek windowPeek)
{
```

```
1824    Rect  theRect;
1825
1826    getGoAwayRect(windowPeek,&theRect);
1827
1828    if(gColorQuickDrawPresent && gColourDisplay)
1829      PlotCIcon(&theRect,gCloseEnabledHdl);
1830    else
1831    {
1832      EraseRect(&theRect);
1833      PenSize(1,1);
1834      FrameRect(&theRect);
1835    }
1836  }
1837
1838  // ##################################################################### drawGoAwayBoxPressed
1839
1840  void  drawGoAwayBoxPressed(WindowPeek windowPeek)
1841  {
1842    Rect  theRect;
1843
1844    getGoAwayRect(windowPeek,&theRect);
1845
1846    if(gColorQuickDrawPresent && gColourDisplay)
1847      PlotCIcon(&theRect,gClosePressedHdl);
1848    else
1849    {
1850      PenSize(2,2);
1851      FrameRect(&theRect);
1852    }
1853  }
1854
1855  // ##################################################################### getGoAwayRect
1856
1857  void  getGoAwayRect(WindowPeek windowPeek,Rect *theRect)
1858  {
1859    getStructRect(windowPeek,theRect);
1860
1861    if(gColorQuickDrawPresent && gColourDisplay)
1862    {
1863      theRect->top += 3;
1864      theRect->left += 3;
1865      theRect->bottom = theRect->top + 8;
1866      theRect->right = theRect->left + 8;
1867    }
1868    else
1869    {
1870      theRect->top += 2;
1871      theRect->left += 2;
1872      theRect->bottom = theRect->top + 7;
1873      theRect->right = theRect->left + 7;
1874    }
1875  }
1876
1877  // ##################################################################### getContentRect
1878
1879  void  getContentRect(WindowPeek windowPeek,Rect *theRect)
1880  {
1881    GrafPtr oldPort;
1882
1883    *theRect = windowPeek->port.portRect;
1884
1885    GetPort(&oldPort);
1886    SetPort((GrafPtr) windowPeek);
1887
1888    LocalToGlobal((Point *) &(theRect->top));
1889    LocalToGlobal((Point *) &(theRect->bottom));
1890
1891    SetPort(oldPort);
1892  }
1893
1894  // ##################################################################### getStructRect
1895
1896  void  getStructRect(WindowPeek windowPeek,Rect *theRect)
1897  {
1898    getContentRect(windowPeek,theRect);
1899
1900    if(gColorQuickDrawPresent && gColourDisplay)
```

```
  {
    InsetRect(theRect,-4,-4);
    theRect->top -= 10;
  }
  else
  {
    theRect->top -= 10;
    InsetRect(theRect,-1,-1);
  }
}

// ############################################################################## syncPorts

void  syncPorts(void)
{
  GrafPtr   bwPort;
  CGrafPtr  colorPort;

  GetWMgrPort(&bwPort);
  GetCWMgrPort(&colorPort);
  SetPort((GrafPtr) colorPort);

  BlockMoveData(&bwPort->pnLoc,&colorPort->pnLoc,10);
  BlockMoveData(&bwPort->pnVis,&colorPort->pnVis,14);

  PenPat(&bwPort->pnPat);
  BackPat(&bwPort->bkPat);
}

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

Demonstration Program Comments

When this program is run, the user should firstly open two or three document windows. The user may then observe the following behaviour:

- When a document window, the Tools floating window, or the Colours floating window is clicked, one, two, or three beeps are played as proof that the program "knows" which window the mouse-down occurred in.
- When a non-active document window is clicked, advisory text is drawn at the bottom of the windows being deactivated and activated as proof that the program "knows" which windows to activate and deactivate.
- 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… alert box or the Find… dialog box 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 About… alert box or the Find… dialog box is invoked.
  - The floating windows are hidden when the program is sent to the background and shown again when the program is brought to the foreground.
  - The floating windows may be toggled between the hidden and shown state by choosing the relevant item in the Floaters menu. In addition, they may be hidden by clicking their close boxes.

Floaters.h

This is the first demonstration program with an application header file. Because the source code is divided into two files (FloaterDemo.c and FloatRoutines.c), all #includes, #defines, #typedefs, and function prototypes have been placed in Floaters.h, which is included by both FloaterDemo.c and FloatRoutines.c.

#define

Lines 62-72 establish constants relating to menu IDs and menu item numbers. Lines 74-84 establish constants relating to menu bar, window, alert, dialog, picture, and sound resources. Lines 86-87 establish constants which will be used to identify a particular window as being of the document type or the floating type. Line 89 defines MAXLONG as the maximum possible long value.

#typedef

As will be seen, a document record will be created for all windows, including the two floating windows, and a pointer to the appropriate application-defined window activation routine will be assigned to the second field of each document record. Line 93 defines ActivateProcPtr as a pointer to a Pascal function that takes a WindowPtr and a Boolean as parameters and returns nothing.

Lines 95-100 define a data type for a document record. A document record, although somewhat of a misnomer in the case of the floating windows, will be created for both floating and document windows. The first field will be assigned a value represented by the constant kDocumentKind (document windows) or kFloatingKind (floating windows). The second field will be assigned a pointer to the window activation routine to be called in respect of each of these windows/window types. The third field, which will keep track of floating window visibility prior to a switch to the background, will be set and read by the application-defined functions which handle suspend and resume events.

FloaterDemo.c

Global Variables

gColorQuickDraw will be set to true if Color QuickDraw is present. gColourDisplay will be set to true if the pixel depth of the main device is greater than 1. gDone controls program termination. gInBackground relates to foreground/background switching. gNumberDocWindowsOpen will keep track of the number of document windows open at any one time. The remaining two global variables will be assigned pointers to the two floating windows.

main

The main function firstly initialises the system software managers (Line 192). At Lines 196-205, the global variable gColorQuickDrawPresent is set to true if Color QuickDraw is present and, if Color QuickDraw is present, gColourDisplay is set to true if the pixel depth of the main device is greater than 1. Lines 209-219 set up the menus and Lines 223-226 opens the two floating windows and sets a checkmark in their respective Floaters menu items. The main event loop is entered at Line 230.

doOpenFloatingWindows

doOpenFloatingWindows opens the two floating windows.

Lines 259-260 set a variable which will control which 'PICT' resources (colour or black-and-white) will be loaded depending on whether Color QuickDraw is present or not.

Line 262 gets the address of the application-defined routine for activating/deactivating the Tools floating window. Line 263 opens the Tools floating window by calling the special application-defined function for opening windows in a floating windows environment. At Line 270, the appropriate 'PICT' resource for the Tools window is loaded. The call to SetWindowPic at Line 271 stores the handle to the picture record in the windowPic field of the window record, meaning that the Window Manager will draw the picture in the window instead of generating update events for it.

Lines 273-282 repeat this process for the Colours window. Line 275 re-highlights the Tools window. Both windows are now highlighted and active.

eventLoop

eventLoop is the main event loop.

When a non-null event is returned, and if the event does not belong to a dialog window (Line 301), the program's event handler is called (Line 302). If the event belongs to a dialog window (Line 303), DialogSelect is called at Line 305 to process the event. When DialogSelect returns, Lines 306-307 dispose of the dialog.

doEvents

doEvents performs initial event handling.

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.

doMouseDown

doMouseDown further processes mouse-down events.

If the mouse-down was in the content region of a window (Line 371), the following occurs. If the front window is not a dialog window (Line 373), an application-defined function is called at Line 374 to play one, two, or three beeps depending on whether the window clicked was a document window, the Tools window, or the Colours window. If the front window is a modal dialog and the window clicked is not the modal dialog window (Line 375), the system alert sound is played (Line 376); otherwise, if the window clicked is not the front window, the special application-defined function which replaces SelectWindow in a floating window environment is called (Lines 377-379).

If the mouse-down was in the title bar of a window (Line 382), the following occurs. If the front window is a modal dialog and the mouse-down was not within the dialog's window, the system alert sound is played (Lines 383-385); otherwise, the special application-defined function which replaces DragWindow in a floating window environment is called (Lines 386-387).

If the mouse-down was in the close box (Line 390), and if TrackGoAway returns true (Line 391), the following occurs. A value representing the window type (floating or document) is retrieved from the window's document record (Lines 393-394). If the window is a floating window (Line 395), the special application-defined function which replaces HideWindow in a floating window environment is called (Line 397) and the menus are adjusted to remove the checkmark from the relevant Floaters menu item (Line 398). If the window is a document window, the application-defined function which closes document windows is called (Lines 400-401).

The handling of mouse-downs in the grow box and the zoom box (Lines 406-428) is as for handling in a non-floating windows environment. Note that the third parameter in the call to ZoomWindow at Line 424 must be false so that the window is not brought to the front.

doUpdate

doUpdate further processes update events. Recall that, because of the SetWindowPic calls at Lines 271 and 282, the floating windows will not receive update events. Accordingly, the only windows redrawn by this function are document windows (Line 442).

doOSEvent

doOSEvent further processes Operating System events.

In the case of a resume event, the special application-defined function which handles resume events in a floating windows environment is called (Line 457). In the case of a suspend event, the special application-defined function which handles suspend events in a floating windows environment is called (Line 463). As usual, the global variable gInBackground is set appropriately although, in this particular demonstration program, it actually has no part to play.

doAdjustMenus

doAdjustMenus adjusts the menus, ensuring that the appropriate disabling is effected when the front window is a modal dialog, that the Close item in the File menu is enabled only if at least one document window is open, and that the items in the Floaters menu are only checkmarked when the relevant floating window is showing.

doMenuChoice

doMenuChoice performs initial menu choice handling.

If the About… item in the Apple menu is chosen, the special application-defined function which performs floating window and document window deactivation in a floating windows environment is called before the alert box is invoked (Lines 529-532). When the alert box is dismissed, the special application-defined function which performs floating window and document window activation is called (Line 534).

doFileMenu

doFileMenu further processes File menu choices.

doFloatersMenu

doFloatersMenu further processes Floater menu choices.

Lines 586-589 assign the pointer to the floating window associated with the chosen item to a variable. Line 591 determines whether that floating window is currently visible. If it is not visible, the special application-defined function which replaces ShowWindow in a floating windows environment is called (Lines 593-594); otherwise, the special application-defined function which replaces HideWindow in a floating windows environment is called (Lines 595-596).

Line 598 adjusts the menus to include/remove the checkmark in/from the relevant Floaters menu item, as appropriate.

doOpenDocWindow

doOpenDocWindow is called in response to a choice of the Open item in the File menu.

If the number of document windows currently open is three, the system alert sound is played and the function returns (Lines 608-609); otherwise, Line 603 gets a pointer to the application-defined function for activating/deactivating document windows, the special application-defined functions which open windows and replace ShowWindow in a floating windows environment are called (Lines 613-614) and the global variable which keeps track of the number of open windows is incremented (Line 615).

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.

Line 625 attempts to get a pointer to the front document window. If NULL is returned by findFrontNonFloatingWindow, no document windows are open. If a non-NULL value is returned, the special application-defined function which closes windows in a floating windows environment is called (Lines 627-628). Line 630 decrements the global variable which keeps track of the number of open windows.

doDrawDocWindowContent

doDrawDocWindowContent is called when an update event is received. It simply retrieves three strings containing advisory text and draws them in the window receiving the update message.

doProvingBeeps

doProvingBeeps is called in the event of a mouse-down in the content region of a non-dialog window. It plays a sound one, two, or three times depending on whether the mouse-down was in a document window, the Tools floating window, or the Colours floating window. (A special sound is used so that these particular "beeps" may distinguished from the normal system alert sound.)

doOpenFindDialog

doOpenFindDialog responds to the choice of the Find… item in the File menu. Before opening the dialog (Line 677), a call is made to the special application-defined function which handles window deactivation in a floating windows environment. The menus are then adjusted as appropriate in the presence of an open modal dialog.

doDisposeFindDialog

doDisposeFindDialog is called when the user dismisses the Find dialog. After closing the dialog (Lines 688-689), a call is made to the special application-defined function which handles window activation in a floating windows environment (Line 691). The menus are then re-adjusted (Line 693).

docActivateHandler

docActivateHandler is the first of three window activation routines.

In a real application docActivateHandler would complete the window activation/deactivation process for document windows. In this demonstration, all that is done is to briefly display the text "Activating" or "Deactivating" in the bottom of the window according to the value in the becomingActive parameter.

toolsActivateHandler

toolsActivateHandler is the activate routine for the Tools floating window. It completes the window activation/deactivation process for this window.

Line 734 sets Tools floating window's graphics port as the current graphics port.

Lines 736-742 determine which 'PICT' resource will be loaded by the GetPicture call at Line 744. If the window is being deactivated (which only happens when an alert or modal dialog is invoked), a dimmed version of the picture is loaded, otherwise the normal (bright) version is loaded. In addition, if Color QuickDraw is present, a colour version is loaded, otherwise a black-and-white version is loaded.

After the appropriate 'PICT' resource is loaded (Line 744), a copy is made of the rectangle in the picture record's picFrame field (Line 745) and that rectangle is offset so that the left and top fields are both 0. Lines 747-748 erase the port rectangle if Color QuickDraw is not present. Line 749 draws the picture and Line 750 stores the handle to the picture record in the windowPic field of the window record, meaning that the Window Manager will draw the picture in the window instead of generating update events for it.

coloursActivateHandler

coloursActivateHandler is the activate routine for the Colours floating window. It completes the window activation/deactivation process for this window. It is identical to toolsActivateHandler except in respect of the 'PICT' resources loaded.

invalidateScrollBarArea

invalidateScrollBarArea invalidates those parts of the window's content region which are occupied by the scroll bars.

FloatRoutines.c

FloatWindows.c contains the special application-defined routines required in a floating windows environment, including routines which are called in lieu of the usual calls to GetNewWindow, DisposeWindow, SelectWindow, HideWindow, ShowWindow, and DragWindow.

newGetNewWindow

newGetNewWindow is called in lieu of the normal call to GetNewWindow/GetNewCWindow. When it opens a floating window it brings it to the very front of any existing windows. When it opens a document window specified to be opened in front of existing document windows, and if any floating windows are already open, it will move the new document window immediately behind the last floating window in the list.

Line 825 assigns memory for the window's document record. Lines 827-831 attempt to open a colour or a black-and-white window depending on whether Color QuickDraw is present.

If the window is opened successfully (Line 833), the handle to the document record is assigned to the window record's refCon field (Line 835) and the pointer to the window's activation function is assigned to the activateHandler field of the window's document record (Line 836). If the window is a floating window (Line 838), the windowType field of the window's document record is assigned a value representing that window kind and the window is highlighted (Lines 840-841). If the window is not of the floating kind (Line 843), the windowType field of the window's document record is assigned a value representing the document window kind and, if the window is required to be opened in front of other document windows, the following occurs: Line 848 attempts to get a pointer to the last floating window in the window list; if any floating windows are currently open, the document window is moved behind the last floating window (Lines 850-851), otherwise the window is brought to the front of all windows (Lines 852-853).

If the window was not successfully opened, the document record is disposed of (Lines 857-858). Line 860 returns the result of the call to GetNewWindow/GetNewCWindow.

newDisposeWindow

newDisposeWindow is called in lieu of the normal calls to DisposeWindow. It ensures that, when a document window is closed, the next document window in the list (if any) is activated.

If the specified window (which will invariably be a document window) is visible (Line 867), a call is made to the function which replaces HideWindow so that the next document window in the list (if any) is activated (Line 868). Line 969 then removes the window from the screen and the window list, Line 870 disposes of the window's document record, and Line 871 disposes of the window's window record.

newSelectWindow

newSelectWindow is called in lieu of the normal call to SelectWindow. It brings a window (floating or document) as far forward in the window list as it should come when the user clicks it. 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).

If the window clicked is of the floating kind (Line 882), Line 884 records that fact and Line 885 gets the pointer to the current front window, which will be a floating window. If the window clicked was a document window (Line 887), Line 889 records that fact, Line 890 gets a pointer to the first document window in the list, and Line 891 gets a pointer to the last floating window in the window list.

If the window clicked is not the current front window in either the floating or document window sections of the window list (Line 894), and if the window clicked is a floating window (Line 896), that window is brought to the very front of the list (Line 897). If the window clicked is a document window (Line 898), and if their are no floating windows (Line 900), SelectWindow is called as in a non-floating-windows environment, and with the same window activation/deactivation effects. If, however, one or more floating windows have been opened (Line 902), an application-defined function is called to effect deactivation of the front document window, SendBehind is called to move the clicked window to immediately behind the last floating window, and an application-defined function is called to effect activation of the clicked document window.

newHideWindow

newHideWindow is called in lieu of the normal call to HideWindow. It hides the specified window. As with HideWindow, if the frontmost window is to be hidden, it is placed behind the window immediately behind it so that, when it is shown again, it will no longer be frontmost. This is also true of document windows even if floating windows are currently visible.

If the specified window is not visible, the function returns without doing anything (Lines 921-922).

Line 924 gets a pointer to the frontmost window. If this window is not a floating window, a variable is set to record that fact (Lines 925-926).

Line 928 gets a pointer to the first document window.

Line 930 hides the specified window without affecting the front-to-back ordering of the windows.

If the newly hidden window is the front floating window (Line 932), and if the next window in the list (if any) is a floating window (Lines 934-936) the hidden window is moved behind that window (Lines 938-940). (This latter is achieved by setting the nextWindow fields in the two floating window records appropriately and then assigning the pointer to the new front floating window's window record to the low-memory global WindowList, which contains a pointer to the first window record in the window list.)

If the newly hidden window is not the front floating window (Line 943), if it is the front document window (Line 945), and if there is another document window behind it (Lines 947-949), the following occurs. Lines 951-952 set the nextWindow fields of the two window records appropriately so as to swap their positions in the list. If one or more floating windows are open (Line 954-955), Line 956 sets the nextWindow field of the last floating window's window record to the new front document window; otherwise, the low memory global WindowList is assigned the pointer to the new front document window's window record (Lines 957-958). The new front document window is then activated (Line 960).

newShowWindow

newShowWindow is called in lieu of the normal call to ShowWindow. If the specified (hidden) window is the frontmost document window, the activation routine for the window (if any) behind it is called, and the specified window is shown, highlighted, and its activation routine called. If the specified (hidden) window is a floating window, the window is shown and, unless a modal dialog is present, highlighted, and its activation routine called.

If the specified window is currently visible, the function returns without doing anything (Lines 976-977).

Line 979 gets the type of the specified window. If the (currently hidden) window is a document window (Line 981), and if that window is the front document window (Lines 983-984), the following occurs. If there is a document window behind the specified window (Line 986), that window is deactivated (Line 987). The hilited field of the specified window's window record is then set to true and the variable windowIsInFront is set to true. This latter will eventually (Lines 1012-1016) cause the window's activation function to be called.

If the specified (currently hidden) window is a floating window, Line 995 gets a pointer to the front non-floating window. If at least one non-floating window is open, if that window is the absolute frontmost window, and if it is a modal dialog, the specified (floating) window's hilited field is set to false; otherwise, the specified (floating) window's hilited field is set to true and the variable windowIsInFront is set to true. This latter will eventually (Lines 1012-1016) cause the window's activation function to be called.

Regardless of what has gone on to this point, Line 1010 shows the specified window without affecting the front-to-back ordering of the windows.

If the variable windowIsInFront has been set to true by the foregoing, the pointer to the application-defined window activation/deactivation function for the specified window is retrieved from the window's window record and that function is called to complete the activation process, the true parameter advising that function that activation-related actions, as opposed to deactivation-related actions, are to be performed (Lines 1012-1016).

newDragWindow

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

WaitMouseUp (Line 1037) tests whether the mouse button has remained down since the last mouseDown event. If it has, the following occurs.

Lines 1039-1043 adjust the top of the dragging rectangle so that it is below the menu bar. Lines 1045-1047 save the current graphics port, and set the window manager port as the current graphics port. Line 1049 sets the clipping region to the region below the menu bar.

Lines 1051-1053 check whether the Command key is down and, if so, set a variable accordingly. If the window is a document window and the Command key is not down, the ClipAbove call at Line 1049 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 a document window and the Command key is down, the ClipAbove call at Line 1060 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.)

Lines 1063-1064 create a region to drag, specifically, the structure region of the specified window. This is passed as a parameter to the call to DragGrayRgn at Line 1066, 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, Line 1068 sets the port saved at Line 1045 as the current graphics port.

DragGrayRgn returns a long. 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 (Line 1070), the value in both words is retrieved at Lines 1072-1073. If the mouse was not outside the slop rectangle (Line 1075), the new horizontal and vertical global coordinates are calculated (Lines 1079-1080) and passed as a parameter to the MoveWindow call at Line 1082, which moves the window to the new location without bringing it to the front.

If the Command key was not down during the drag (Line 1086), the call to newSelectWindow at Line 1087 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).

handleSuspendEvent

handleSuspendEvent hides any floating windows and deactivates the frontmost document window. It is called when the application receives a suspend event.

Line 1100 gets the pointer to the front window's window record from the low-memory global WindowList. If this is not a pointer to a floating window (Line 1102), the function simply returns.

The first time through the do-while loop entered at Line 1105, the current visibility status of the first window in the list (which must be a floating window) is saved to the wasVisible field of its window record (Lines 1107-1108). If the window is visible, it is hidden without affecting the front-to-back ordering of the open windows (Lines 1109-1110). Line 1111 attempts to get a pointer to the next window in the list. If there is another window in the list, and if it is a floating window, the same process is repeated. The loop exits only when the visibility status of all floating windows has been saved and those windows have been hidden.

If there are any document windows, the frontmost document window is then deactivated and a call is made to DrawGrowIcon (Lines 1115-1120).

> Note that the call to DrawGrowIcon at Line 1118 should be removed if your do not require your document windows to be resizeable.

handleResumeEvent

handleResumeEvent shows all floating windows which were visible when the application was sent to the background, and activates the front document window. It is called when the application receives a resume event.

Line 1130 gets the pointer to the front window's window record from the low-memory global WindowList. If this is not a pointer to a floating window (Line 1132), the function simply returns.

The first time through the do-while loop entered at Line 1135, the visibility status of the first window in the list, which was saved when the application was sent to the background, is retrieved from the wasVisible field of its window record (Line 1137). If the window was visible, it is shown without affecting the front-to-back ordering of the open windows and then activated (Lines 1138-1142). Line 1143 attempts to get a pointer to the next window in the list. If there is another window in the list, and if it is a floating window, the same process is repeated. The loop exits only when all of the floating windows which were visible when the application was sent to the background have been shown and activated.

If there are any document windows, the frontmost document window is then activated and a call is made to DrawGrowIcon (Lines 1147-1152).

> Note that the call to DrawGrowIcon at Line 1150 should be removed if your do not require your document windows to be resizeable.

deactivateFloatsAndFirstDocWin

deactivateFloatsAndFirstDocWin unhighlights any visible floating windows and the frontmost document window, and then calls the activation function for each window to complete the deactivation process. It is called immediately before a modal dialog or alert box is invoked.

Line 1154 gets a pointer to the frontmost window. Line 1164 attempts to get a pointer to the first document window. If at least one document window exists (Line 1165), Line 1166 attempts to get a pointer to the second document. The variable secondDocumentWindowPtr now contains either a pointer or NULL. The while loop entered at Line 1169 walks the window list up to and including the first document window, deactivating all visible windows.

activateFloatsAndFirstDocWin

activateFloatsAndFirstDocWin highlights and activates those windows which were visible, highlighted and activated before deactivateFloatersAndFirstDocWin was called. It is thus called immediately after an alert or modal dialog box is dismissed. However, if the application is in the background when this function is called (such as when a movable modal progress dialog was up and then disappears), this function does not perform those actions. Instead, it calls handleSuspendEvent to hide any visible floating windows.

Lines 1191-1195 determine whether this program is in the background. If it is, any visible floating windows are hidden (Lines 1197-1198).

If the program is not in the background (Line 1199), Lines 1201-1211 activate all floating windows and the first document window, using the same list-walking methodology as deactivateFloatersAndFirstDocWin.

deactivateWindow, activateWindow, and highlightAndActivateWindow

deactivateWindow and/or activateWindow are called from newSelectWindow, newHideWindow, newShowWindow, handleSuspendEvent, handleResumeEvent, deactivateFloatsAndFirstDocWin, and 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 (Line 1236), retrieves the pointer to the window's activation routine from the window's document record (Line 1237), and calls that routine (Line 1238).

findFrontNonFloatingWindow, Etc

findFrontNonFloatingWindow is the first of those functions which support the main FloatRoutines.c functions already described. It returns a pointer to the first visible window in the window list that is not a floating window.

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

isWindowModal determines whether a window is modal.

getWindowList returns the WindowPeek of the first window in the window list, which is stored in the low memory global WindowList.

setWindowList sets the value in the low-memory global WindowList.

getNextWindow returns the value in the nextWindow field of the specified window.

setNextWindow sets the value in the nextWindow field of the specified window.

getWasVisible returns the value in the wasVisible field of the specified window's document record.

setWasVisible sets the value in the wasVisible field of the specified window's document record.

getWindowKind returns the value in the windowType field of the specified window's document record, or 0 if the window is a dialog.

setWindowKind sets the value in the windowType field of the specified window's document record.

getWindowVisible returns the value in the visible field of the specified window's window record.

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

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

setWindowHilite sets the hilited field of the specified window to the value received in the second parameter.

getActivateHandler returns the value in the activateHandler field of the specified window's window record.

setActivateHandler sets the value in the activateHandler field of the specified window's window record.

WDEF.c

WDEF.c defines the window definition function used by the Tools and Colours floating windows. In the demonstration program, the 'WIND' resources for the floating windows specify a window definition ID of 2048, meaning that the 'WDEF' resource with an ID of 128 is to be used.

#define

Lines 1444-1446 establish constants representing the resource Ids for three colour icons, one for the close box in its normal state, one for the close box in its pressed state, and one which will be used to draw the checkered pattern in the window's title bar.

Global Variables

gColourQuickDrawPresent will be set to true if Color QuickDraw is present. gColourDisplay will be set to true if the pixel depth of the main device is greater than 1. The elements of gBlackPattern will be assigned values representing a black pattern. The three global variables at Lines 1453-1455 will be assigned handles to the three colour icons. The global variables at Lines 1456-1465 are assigned colours according with the various shades of gray specified in the document Apple Grayscale Appearance for System 7.5 published by Apple Computer, Inc. gToggle will be used to specify whether the close box is to be drawn in the normal state or the pressed state.

main

The main function receives the four parameters passed to it by the Window Manager and switches according to the content of the message parameter. The WDEF responds to the following messages: wDraw, wHit, wCalcRgns, and wNew,

Line 1492 has to do with accessing the WDEF's global variables. (See Setting up globals in 68K code resources in the Writing Code Resources section of Chapter 4 (Creating Code Resource Projects) in the CodeWarrior manual Targeting Mac OS.)

Line 1494 saves the current pen location, size, mode and pattern. Line 1495 saves the current graphics port. If Color QuickDraw is present, the WDEF-defined function syncPorts is called (Lines 1497-1498). (As will be seen, the global variable gColorQuickDrawPresent is set in the WDEF's response to the wNew message.)

Line 1500 initialises the variable which contains the value to be returned by the WDEF.

Lines 1502-1520 switch according to the value received in the message parameter. Note that, at Lines 1508-1511, action is taken in response to the wDraw message only if the specified window is currently visible.

Lines 1522-1523 restore the saved pen state and graphics port. Line 1525 restores the A4 register's value saved at Line 1492.

Line 1527 returns the value in the variable result, which will be 0 except when the wHit message is responded to, in which case it will be either wInContent, wInGoAway, wInDrag, or wNoHit.

doInitMessage

doInitMessage is called when the wNew message is received.

At Lines 1539-1548, the global variable gColorQuickDrawPresent is set to true if Color QuickDraw is present and, if Color QuickDraw is present, gColourDisplay is set to true if the pixel depth of the main device is greater than 1.

Lines 1550-1551 initialise the global variable gBlackPattern.

Lines 1553-1555 load the three colour icons.

Line 1557 initialises the gToggle global variable, which will be used in toggling the close box between the normal and highlighted states.

doDrawMessage

doDrawMessage is called when the wDraw message is received.

Line 1564 clears the high word of the param parameter because, in the case of the wDraw message, the high word may contain undefined data.

Lines 1566-1582 switch according to the value in the low word of the param parameter. This value will call for either the entire window frame to be drawn (Lines 1603-1568) or the close box to be toggled to its opposite state (Lines 1575-1577). Note that, if Color QuickDraw is present and the pixel depth of the main device is greater than 1, a colour drawing function is called, otherwise a black-and-white drawing function is called.

doHitMessage

doHitMessage is called when the wHit message is received. Its purpose is to determine if and where a mouse-down occurred within the floating window.

Lines 1591-1592 extract the location of the mouse-down from the param parameter, which is in global coordinates.

If the mouse-down was in the content region, wInContent is returned (Lines 1594-1595) and the function exits. If not, Line 1596 checks whether the mouse-down was in the structure region. If it was, and if the window has a close box (Line 1598), the close box rectangle is retrieved (Line 1600). If the mouse-down was in the close box, wInGoAway is returned and the function exits (Lines 1601-1602), otherwise wInDrag is returned and the function exits (Line 1605).

If none of these checks prove positive, wNoHit is returned (Line 1608).

Note that there is no necessity to check that the window is active at the beginning of this function (by checking the window record's hilited field) because all visible floating windows are always active.

doCalcRgnsMessage

doCalcRgnsMessage is called when the wCalcRgns message is received. It calculates the window's content and structure regions and assigns the result in the window record's contRgn and strucRgn fields.

Line 1620 calls a WDEF-defined function to get a rectangle equivalent to the window's port rectangle converted to global coordinates. Line 1621 sets the window's content region to equate to this rectangle.

Line 1623 calls a WDEF-defined function to get a rectangle which is equivalent to the content rectangle expanded by a certain number of pixels. This rectangle, which is in global coordinates, defines the structure region less the window's drop shadow. As a first step, Line 1624 sets the window's structure region to equate to this rectangle. The rectangle is then offset 1 pixel down and to the right (Line 1625) and the top and left values are further increased (by one pixel) at Lines 1626-1627. Line 1628 then turns this rectangle into a region and Line 1629 combines this region with the region in the window record's strucRgn field so that this field now contains a region which includes the window's drop shadow.

drawWindowColour

drawWindowColour draws the window frame in accordance with the Apple Grayscale Appearance specification for utility windows. This function is called only if Color QuickDraw is present and the main device pixel depth is greater than 1.

Lines 1643-1644 save the current foreground and background colours. Lines 1645-1647 set the pen size, pattern, and transfer mode.

Lines 1650-1713 execute only if the window is currently active (Line 1649). These lines draw a black frame one pixel outside the content rectangle (Lines 1651-1655) and then base all drawing on the structure rectangle (Line 1657). Lines 1659-1662 paint the title bar to the right of the close box rectangle. Lines 1664-1671 use the checkered colour icon to draw the checkered pattern in the title bar. Lines 1673-1674 use the colour icon for the close box in the normal state to draw the close box. Lines 1676-1713 then draw the remainder of the window frame.

Lines 1717-1733 execute only if the window is currently inactive (Line 1715), and draw the window frame in the inactive state.

Lines 1736-1737 restore the foreground and background colours saved at Lines 1643-1644.

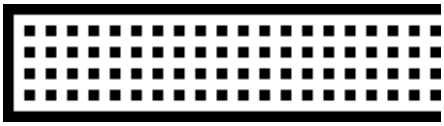
drawWindowMono

drawWindowMono draws the window in black-and-white in accordance with the de facto standard appearance for black-and-white floating windows as seen in many commercial applications. This function is called only if Color QuickDraw is not present or, if Color QuickDraw is present, the main device pixel depth is 1.

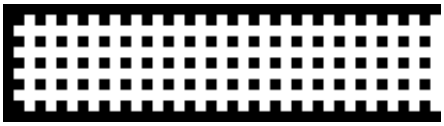
Lines 1745-1751 set the pen size, pattern, and transfer mode.

Lines 1754-1798 execute only if the window is currently active. These lines draw a black frame one pixel outside the content rectangle (Lines 1758-1761) and then base all drawing on the structure rectangle (Line 1763). Lines 1765-1768 frame the structure rectangle and add the drop shadows. Lines 1770-1772 erase the title bar area. Lines 1774-1793 set the pen pattern to a checkered pattern before drawing the title bar. Line 1795 restores the pen pattern to a black pattern and Lines 1797-1798 draw the close box.

Some further explanation of Lines 1774-1788 is necessary. The drawing is taking place in the Window Manager port, which is in global coordinates. This means that any patterns drawn will be aligned to the global origin. Thus, if the pen pattern is simply set to the system pattern number 24, the following will be the result:



1 — TITLE BAR DRAWN WHEN LOCATED AT "EVEN" WINDOW MANAGER PORT COORDINATES



2 — TITLE BAR THEN DRAWN WHEN LOCATED 1 PIXEL FURTHER DOWN AND TO THE RIGHT

The solution to this problem is to change the pattern depending on whether the window is currently located at an even window manager port coordinate or an odd window manager port coordinate. Lines 1777-1785 are central in this respect.

If the window is currently inactive (Line 1670), the title bar is simply erased (Lines 1802-1804).

toggleGoAway

toggleGoAway is called when the wDraw message is received with wInGoAway in the param field. It first reverses the value in the Boolean global gToggle (Line 1812) and then calls the appropriate WDEF-defined function to draw the close box in the normal state or the highlighted state, depending on the value in the gToggle variable (Lines 1814-1817).

drawGoAwayBox

drawGoAwayBox draws the close box in its normal state, either in colour (Lines 1828-1829) or black-and-white if it is not (Lines 1830-1835).

drawGoAwayBoxPressed

drawGoAwayBoxPressed draws the close box in its highlighted state, either in colour or black-and-white.

getGoAwayRect

getGoAwayRect sets the fields of the specified rectangle so as to correctly describe the close box rectangle depending on whether the close box is to be drawn in colour or black-and-white.

getContentRect

getContentRect sets the fields of the specified rectangle to equate to the window's port rectangle expressed in global coordinates.

getStructRect

getStructRect sets the fields of the specified rectangle to equate to the window's port rectangle (expressed in global coordinates) expanded by a number of pixels according to whether the window frame is to be drawn in colour or black-and-white..

syncPorts

syncPorts is called only if Color QuickDraw is present.

When the WDEF is called, the current graphics port is set by the Window Manager to the Window Manager port. This port is a basic graphics port, so it does not permit colour drawing. To draw in colour, it is necessary to switch to another port called the Window Manager color port.

In addition to switching to the Window Manager Color Port, and because the Window Manager does not normally keep all of the fields of both ports synchronised with each other, it is necessary to manually synchronise the ports by copying the contents of some of the fields of the basic Window Manager port to the equivalent fields in the Window Manager color port.

Line 1919 gets a pointer to the Window Manager port. Line 1920 gets a pointer to the Window Manager color port and Line 1921 sets that port as the current port.

Line 1923 copies the contents of the Window Manager port's pnLoc, pnSize, and pnMode fields to the same fields in the Window Manager color port. Line 1924 copies the contents of the Window Manager port's pnVis, txFont, txFace, txMode, txSize, and spExtra fields to the same fields in the Window Manager color port. Lines 1926-1927 set the pen pattern and background pattern to the same values they were in the Window Manager port.

Creating the WDEF Resource

To create an WDEF resource from code such as that at Lines 1418-1930, follow the same general procedure as is described for LDEFs in the Demonstration Program Comments at Chapter 18 — Lists and Custom List Definition Functions.