20 Version 1.1

FLOATING WINDOWS AND CUSTOM WINDOW DEFINITION FUNCTIONS

Includes Demonstration Program FloatersPascal

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.

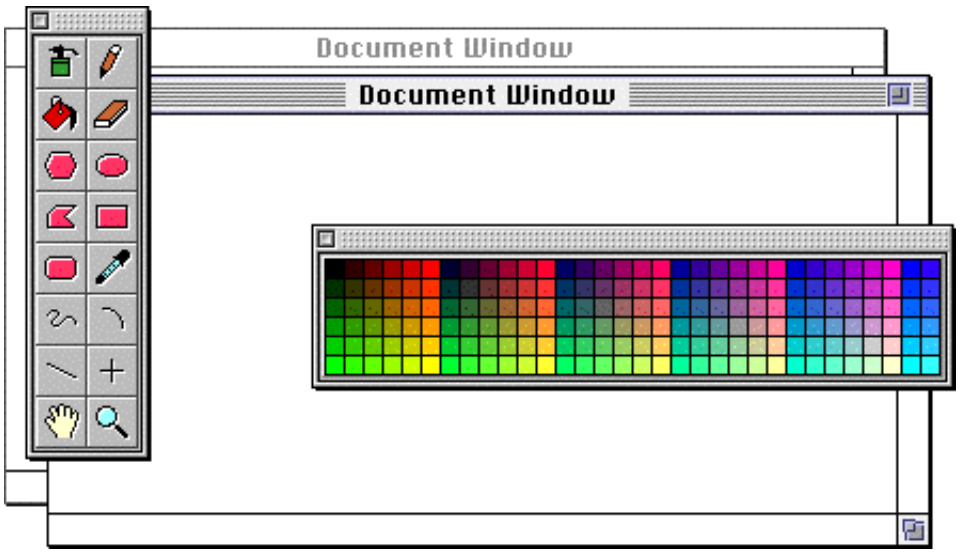


FIG 1 - FLOATING WINDOWS - EXAMPLES

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

```
procedure GetWMgrPort(var wPort: GrafPtr);
procedure GetCWMgrPort(var wMgrCPort: CGrafPtr);
procedure ShowHide(theWindow: WindowRef; showFlag: boolean);
procedure HiliteWindow(theWindow: WindowRef; fHilite: boolean);
procedure BringToFront(theWindow: WindowRef);
procedure SendBehind(theWindow: WindowRef; behindWindow: WindowRef);
procedure MoveWindow(theWindow: WindowRef; hGlobal: integer; vGlobal: integer;
            front: boolean);
procedure CloseWindow(theWindow: WindowRef);
procedure ClipAbove(window: WindowRef);

function  DragGrayRgn(theRgn: RgnHandle; startPt: Point; var limitRect: Rect;
            var slopRect: Rect; axis: integer; actionProc: DragGrayRgnUPP): longint;
```

Demonstration Program

```
{ ############################################################################################
// UFloaters.p
// ############################################################################################ }

unit UFloaters;



interface

{ ……… include the following Universal Interfaces }

uses

  Types, Events, Quickdraw;

{ ……… define and export the following constants }

const

mApple = 128;
 iAbout = 1;
mFile = 129;
 iNew = 1;
 iClose = 4;
 iFind = 11;
 iQuit = 13;
mEdit = 130;
mFloaters = 131;
 iTools = 1;
 iColours = 2;

rMenubar = 128;
rDocWindow = 128;
rToolsWindow = 129;
rColoursWindow = 130;
rAboutAlert = 128;
rFindDialog = 129;
rToolsPict = 128;
rColoursPict = 129;
rToolsPictDim = 130;
rColoursPictDim = 131;
rSound = 8192;

kDocumentKind = 1;
kFloatingKind = 2;

kMaxLong = $7FFFFFFF;

{ ……… user-defined types }

```

```
52  type
53
54  {$IFC PROCTYPE }
55    ActivateProcPtr = PROCEDURE (theWindowPtr : WindowPtr; activate : boolean);
56  {$ELSEC}
57    ActivateProcPtr = ProcPtr;{ PROCEDURE(theWindowPtr : WindowPtr; activate : boolean); }
58  {$ENDC}
59
60  DocRecord = record
61    windowType : integer;
62    activateHandler : ActivateProcPtr;
63    wasVisible : boolean;
64    end;
65
66  DocRecordPtr = ^DocRecord;
67  DocRecordHandle = ^DocRecordPtr;
68
69  { ……exported global variables }
70
71  var
72
73  gColorQuickDrawPresent : boolean;
74  gColourDisplay : boolean;
75  gDone : boolean;
76  gInBackground : boolean;
77  gNumberDocWindowsOpen : integer;
78  gToolsWindowPtr : WindowPtr;
79  gColoursWindowPtr : WindowPtr;
80
81  { ……exported functions and procedures }
82
83    procedure DoInitManagers;
84    procedure DoOpenFloatingWindows;
85    procedure EventLoop;
86
87
88
89  implementation
90
91  { ……include the following Universal Interfaces }
92
93  uses
94
95    Windows, Fonts, Menus, TextEdit, Dialogs, QuickdrawText, Processes, Resources,
96    Memory, TextUtils, ToolUtils, OSUtils, Devices, SegLoad, Sound,
97
98  { ……include the following user-defined units }
99
100   UFloatRoutines;
101
102 { ……procedure and function definitions }
103
104   procedure DoEvents(eventRec : EventRecord); forward;
105   procedure DoMouseDown(eventRec : EventRecord); forward;
106   procedure DoUpdate(eventRec : EventRecord); forward;
107   procedure DoOSEvent(eventRec : EventRecord); forward;
108   procedure DoAdjustMenus; forward;
109   procedure DoMenuChoice(menuChoice : longint); forward;
110   procedure DoFileMenu(menuItem : integer); forward;
111   procedure DoFloatersMenu(menuItem : integer); forward;
112   procedure DoOpenDocWindow; forward;
113   procedure DoCloseDocWindow; forward;
114   procedure DoDrawDocWindowContent; forward;
115   procedure DoProvingBeeps(theWindowPtr : WindowPtr); forward;
116   procedure DoOpenFindDialog; forward;
117   procedure DoDisposeFindDialog; forward;
118   procedure DocActivateHandler(theWindowPtr : WindowPtr; activate : boolean); forward;
119   procedure ToolsActivateHandler(theWindowPtr : WindowPtr; activate : boolean); forward;
120   procedure ColoursActivateHandler(theWindowPtr : WindowPtr;
121         activate : boolean); forward;
122   procedure InvalidateScrollBarArea(theWindowPtr : WindowPtr); forward;
123
124 { ……procedure and function implementations }
125
126 { ########################################################################## DoInitManagers }
127
128 procedure DoInitManagers;
```

```
129
130     begin
131     MaxApplZone;
132     MoreMasters;
133
134     InitGraf(@qd.thePort);
135     InitFonts;
136     InitWindows;
137     InitMenus;
138     TEInit;
139     InitDialogs(nil);
140
141     InitCursor;
142     FlushEvents(everyEvent, 0);
143     end;
144       {of procedure DoInitManagers}
145
146   { ################################################################ DoOpenFloatingWindows }
147
148   procedure DoOpenFloatingWindows;
149
150     var
151     resourceOffset : integer;
152     pictureHdl : PicHandle;
153     theProcPtr : ActivateProcPtr;
154
155     begin
156     resourceOffset := 0;
157
158     if ((gColorQuickDrawPresent = false) or (gColourDisplay = false)) then
159       resourceOffset := 4;
160
161     theProcPtr := ActivateProcPtr(@ToolsActivateHandler);
162     gToolsWindowPtr := NewGetNewWindow(rToolsWindow, WindowPtr(-1), theProcPtr,
163           kFloatingKind);
164     if (gToolsWindowPtr = nil) then
165       begin
166       SysBeep(10);
167       ExitToShell;
168       end;
169
170     pictureHdl := GetPicture(rToolsPict + resourceOffset);
171     SetWindowPic(gToolsWindowPtr, pictureHdl);
172
173     theProcPtr := ActivateProcPtr(@ColoursActivateHandler);
174     gColoursWindowPtr := NewGetNewWindow(rColoursWindow, WindowPtr(-1), theProcPtr,
175           kFloatingKind);
176     if (gColoursWindowPtr = nil) then
177       begin
178       SysBeep(10);
179       ExitToShell;
180       end;
181
182     pictureHdl := GetPicture(rColoursPict + resourceOffset);
183     SetWindowPic(gColoursWindowPtr, pictureHdl);
184
185     HiliteWindow(gToolsWindowPtr, true);
186     end;
187       {of procedure DoOpenFloatingWindows}
188
189   { ################################################################################ EventLoop }
190
191   procedure EventLoop;
192
193     var
194     eventRec : EventRecord;
195       theDialogPtr : DialogPtr;
196     itemHit : integer;
197     ignoredBool : boolean;
198
199     begin
200     gDone := false;
201
202     while not (gDone) do
203       begin
204       if (WaitNextEvent(everyEvent, eventRec, kMaxLong, nil)) then
205         begin
```

```
206         if not (IsDialogEvent(eventRec)) then
207           DoEvents(eventRec)
208         else begin
209           ignoredBool := DialogSelect(eventRec, theDialogPtr, itemHit);
210           if ((itemHit = ok) or (itemHit = cancel)) then
211             DoDisposeFindDialog;
212           end;
213         end;
214       end;
215     end;
216       {of procedure EventLoop}
217
218   { ############################################################################## DoEvents }
219
220   procedure DoEvents(eventRec : EventRecord);
221
222     var
223     charCode : char;
224
225     begin
226     case  (eventRec.what) of
227
228       mouseDown: begin
229         DoMouseDown(eventRec);
230         end;
231
232       keyDown, autoKey: begin
233         charCode := chr(BAnd(eventRec.message, charCodeMask));
234         if (BAnd(eventRec.modifiers, cmdKey) <> 0) then
235           begin
236           DoAdjustMenus;
237           DoMenuChoice(MenuKey(charCode));
238           end;
239         end;
240
241       updateEvt: begin
242         DoUpdate(eventRec);
243         end;
244
245       osEvt: begin
246         DoOSEvent(eventRec);
247         HiliteMenu(0);
248         end;
249       end;
250     end;
251       {of procedure DoEvents}
252
253   { ############################################################################ DoMouseDown }
254
255   procedure DoMouseDown(eventRec : EventRecord);
256
257     var
258     partCode : integer;
259     theWindowPtr : WindowPtr;
260     frontWindowPtr : WindowPtr;
261     docRecordHdl : DocRecordHandle;
262     windowType : integer;
263     newSize : UInt32;
264     growRect : Rect;
265
266     begin
267     partCode := FindWindow(eventRec.where, theWindowPtr);
268
269     case  (partCode) of
270
271       inMenuBar: begin
272         DoAdjustMenus;
273         DoMenuChoice(MenuSelect(eventRec.where));
274         end;
275
276       inSysWindow: begin
277         SystemClick(eventRec, theWindowPtr);
278         end;
279
280       inContent: begin
281         frontWindowPtr := FrontWindow;
282         if (WindowPeek(frontWindowPtr)^.windowKind <> dialogKind) then
```

```
283           DoProvingBeeps(theWindowPtr);
284         if ((IsWindowModal(frontWindowPtr)) and (theWindowPtr <> frontWindowPtr)) then
285           SysBeep(10)
286         else if (theWindowPtr <> frontWindowPtr) then
287           NewSelectWindow(theWindowPtr);
288         end;
289 
290       inDrag: begin
291         frontWindowPtr := FrontWindow;
292         if ((IsWindowModal(frontWindowPtr)) and (theWindowPtr <> frontWindowPtr)) then
293           SysBeep(10)
294         else
295           NewDragWindow(theWindowPtr, eventRec.where, qd.screenBits.bounds);
296         end;
297 
298       inGoAway: begin
299         if (TrackGoAway(theWindowPtr, eventRec.where)) then
300           begin
301           docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));
302           windowType := docRecordHdl^^.windowType;
303           if (windowType = kFloatingKind) then
304             begin
305             NewHideWindow(theWindowPtr);
306             DoAdjustMenus;
307             end
308           else if (windowType = kDocumentKind) then
309             DoCloseDocWindow;
310           end;
311         end;
312 
313       inGrow: begin
314         growRect := qd.screenBits.bounds;
315         growRect.top := 145;
316         growRect.left := 335;
317         newSize := GrowWindow(theWindowPtr, eventRec.where, growRect);
318         if (newSize <> 0) then
319           begin
320           InvalidateScrollBarArea(theWindowPtr);
321           SizeWindow(theWindowPtr, LoWord(newSize), HiWord(newSize), true);
322           InvalidateScrollBarArea(theWindowPtr);
323           end;
324         end;
325 
326       inZoomIn, inZoomOut: begin
327         if (TrackBox(theWindowPtr, eventRec.where, partCode)) then
328           begin
329           SetPort(theWindowPtr);
330           EraseRect(theWindowPtr^.portRect);
331           ZoomWindow(theWindowPtr, partCode, false);
332           InvalRect(theWindowPtr^.portRect);
333           end;
334         end;
335       end;
336     end;
337       {of procedure DoMouseDown}
338 
339   { ############################################################################## DoUpdate }
340 
341   procedure DoUpdate(eventRec : EventRecord);
342 
343     var
344     theWindowPtr : WindowPtr;
345 
346     begin
347     theWindowPtr := WindowPtr(eventRec.message);
348     SetPort(theWindowPtr);
349 
350     BeginUpdate(theWindowPtr);
351     EraseRect(theWindowPtr^.portRect);
352     DoDrawDocWindowContent;
353     DrawGrowIcon(theWindowPtr);
354     EndUpdate(theWindowPtr);
355     end;
356       {of procedure DoUpdate}
357 
358   { ############################################################################ DoOSEvent }
359 
```

```
360 procedure DoOSEvent(eventRec : EventRecord);
361
362   begin
363   case BAnd(BSR(eventRec.message, 24), $000000FF) of
364
365     suspendResumeMessage: begin
366       if (BAnd(eventRec.message, resumeFlag) <> 0) then
367         begin
368         gInBackground := false;
369         HandleResumeEvent;
370         SetCursor(qd.arrow);
371         end
372       else begin
373         gInBackground := true;
374         HandleSuspendEvent;
375         end;
376       end;
377
378     mouseMovedMessage: begin
379       end;
380     end;
381       {of case statement}
382   end;
383     {of procedure DoOSEvent}
384 { ############################################################################ DoAdjustMenus }
385
386 procedure DoAdjustMenus;
387
388   var
389   appleMenuHdl, fileMenuHdl, floatMenuHdl : MenuHandle;
390
391   begin
392   appleMenuHdl := GetMenuHandle(mApple);
393   fileMenuHdl := GetMenuHandle(mFile);
394   floatMenuHdl := GetMenuHandle(mFloaters);
395
396   if (IsWindowModal(FrontWindow)) then
397     begin
398     DisableItem(appleMenuHdl, 1);
399     DisableItem(fileMenuHdl, 0);
400     DisableItem(floatMenuHdl, 0);
401     end
402   else begin
403     EnableItem(appleMenuHdl, 1);
404     EnableItem(fileMenuHdl, 0);
405     EnableItem(floatMenuHdl, 0);
406
407     if (gNumberDocWindowsOpen > 0) then
408       EnableItem(fileMenuHdl, iClose)
409     else
410       DisableItem(fileMenuHdl, iClose);
411     end;
412
413   if (GetWindowVisible(gToolsWindowPtr)) then
414     CheckItem(floatMenuHdl, iTools, true)
415   else
416     CheckItem(floatMenuHdl, iTools, false);
417
418   if (GetWindowVisible(gColoursWindowPtr)) then
419     CheckItem(floatMenuHdl, iColours, true)
420   else
421     CheckItem(floatMenuHdl, iColours, false);
422
423   DrawMenuBar;
424   end;
425     {of procedure DoAdjustMenus}
426
427 { ############################################################################ DoMenuChoice }
428
429 procedure DoMenuChoice(menuChoice : longint);
430
431   var
432   menuID, menuItem : integer;
433   itemName : string;
434   daDriverRefNum : integer;
435   ignoredInt : integer;
436
```

```
437     begin
438     menuID := HiWord(menuChoice);
439     menuItem := LoWord(menuChoice);
440
441     if (menuID = 0) then
442       Exit(DoMenuChoice);
443
444     case (menuID) of
445
446       mApple: begin
447         if (menuItem = iAbout) then
448           begin
449           DeactivateFloatsAndFirstDocWin;
450           ignoredInt := NoteAlert(rAboutAlert, nil);
451           ActivateFloatsAndFirstDocWin;
452           end
453         else begin
454           GetMenuItemText(GetMenuHandle(mApple), menuItem, itemName);
455           daDriverRefNum := OpenDeskAcc(itemName);
456           end;
457         end;
458
459       mFile: begin
460         DoFileMenu(menuItem);
461         end;
462
463       mFloaters: begin
464         DoFloatersMenu(menuItem);
465         end;
466       end;
467         {of case statement}
468
469     HiliteMenu(0);
470     end;
471       {of procedure DoMenuChoice}
472
473   { ############################################################################## DoFileMenu }
474
475   procedure DoFileMenu(menuItem : integer);
476
477     begin
478     case (menuItem) of
479
480       iNew: begin
481         DoOpenDocWindow;
482         end;
483
484       iClose: begin
485         DoCloseDocWindow;
486         end;
487
488       iFind: begin
489         DoOpenFindDialog;
490         end;
491
492       iQuit: begin
493         gDone := true;
494         end;
495       end;
496         {of case statement}
497     end;
498       {of procedure DoFileMenu}
499
500   { ########################################################################## DoFloatersMenu }
501
502   procedure DoFloatersMenu(menuItem : integer);
503
504     var
505     floaterVisible : boolean;
506     floaterPicked : WindowPtr;
507
508     begin
509     if (menuItem = iTools) then
510       floaterPicked := gToolsWindowPtr
511     else if (menuItem = iColours) then
512       floaterPicked := gColoursWindowPtr;
513
```

```
514     floaterVisible := GetWindowVisible(floaterPicked);
515
516     if (floaterVisible = false) then
517       NewShowWindow(floaterPicked)
518     else
519       NewHideWindow(floaterPicked);
520
521     DoAdjustMenus;
522     end;
523       {of procedure DoFloatersMenu}
524
525   { ############################################################################ DoOpenDocWindow }
526
527   procedure DoOpenDocWindow;
528
529     var
530     theProcPtr : ActivateProcPtr;
531     theWindowPtr : WindowPtr;
532
533     begin
534     if (gNumberDocWindowsOpen > 3) then
535       SysBeep(10)
536     else begin
537       theProcPtr := ActivateProcPtr(@DocActivateHandler);
538       theWindowPtr := NewGetNewWindow(rDocWindow, WindowPtr(-1), theProcPtr,
539             kDocumentKind);
540       NewShowWindow(theWindowPtr);
541       gNumberDocWindowsOpen := gNumberDocWindowsOpen + 1;
542       end;
543     end;
544       {of procedure DoOpenDocWindow}
545
546   { ############################################################################ DoCloseDocWindow }
547
548   procedure DoCloseDocWindow;
549
550     var
551     theWindowPtr : WindowPtr;
552
553     begin
554     theWindowPtr := FindFrontNonFloatingWindow;
555
556     if (theWindowPtr <> nil) then
557       begin
558       NewDisposeWindow(theWindowPtr);
559       gNumberDocWindowsOpen := gNumberDocWindowsOpen - 1;
560       end;
561     end;
562       {of procedure DoCloseDocWindow}
563
564   { ################################################################## DoDrawDocWindowContent }
565
566   procedure DoDrawDocWindowContent;
567
568     var
569     theString : string;
570     a : integer;
571
572     begin
573     for a := 1 to 3 do
574       begin
575       GetIndString(theString, 128, a);
576       MoveTo(20, a*30);
577       DrawString(theString);
578       end;
579     end;
580       {of procedure DoDrawDocWindowContent}
581
582   { ############################################################################ DoProvingBeeps }
583
584   procedure DoProvingBeeps(theWindowPtr : WindowPtr);
585
586     var
587     soundHdl : Handle;
588     repeats, a : integer;
589     ignoredErr : OSErr;
590
```

```
591     begin
592     soundHdl := GetResource('snd ', rSound);
593
594     if (theWindowPtr = gToolsWindowPtr) then
595       repeats := 2
596     else if (theWindowPtr = gColoursWindowPtr) then
597       repeats := 3
598     else
599       repeats := 1;
600
601     for a := 0 to (repeats - 1) do
602       ignoredErr := SndPlay(nil, SndListHandle(soundHdl), false);
603
604     ReleaseResource(soundHdl);
605     end;
606       {of procedure DoProvingBeeps}
607
608   { ############################################################################ DoOpenFindDialog }
609
610   procedure DoOpenFindDialog;
611
612     var
613     findDialogPtr : DialogPtr;
614
615     begin
616     DeactivateFloatsAndFirstDocWin;
617     findDialogPtr := GetNewDialog(rFindDialog, nil, WindowPtr(-1));
618
619     DoAdjustMenus;
620     end;
621       {of procedure DoOpenFindDialog}
622
623   { ########################################################################## DoDisposeFindDialog }
624
625   procedure DoDisposeFindDialog;
626
627     var
628     findDialogPtr : DialogPtr;
629
630     begin
631     findDialogPtr := FrontWindow;
632     DisposeDialog(findDialogPtr);
633     ActivateFloatsAndFirstDocWin;
634
635     DoAdjustMenus;
636     end;
637       {of procedure DoDisposeFindDialog}
638
639   { ############################################################################ DocActivateHandler }
640
641   procedure DocActivateHandler(theWindowPtr : WindowPtr; activate : boolean);
642
643     var
644     oldPort : GrafPtr;
645     finalTicks : longint;
646     theRect : Rect;
647
648     begin
649     GetPort(oldPort);
650     SetPort(theWindowPtr);
651
652     DrawGrowIcon(theWindowPtr);
653
654     MoveTo(20, 120);
655     if (activate) then
656       DrawString('Activating')
657     else
658       DrawString('Deactivating');
659
660     Delay(30, finalTicks);
661
662     SetRect(theRect, 20, 105, 100, 125);
663     EraseRect(theRect);
664
665     SetPort(oldPort);
666     end;
667       {of procedure DocActivateHandler}
```

```
668
669 { ###################################################################### ToolsActivateHandler }
670
671 procedure ToolsActivateHandler(theWindowPtr : WindowPtr; activate : boolean);
672
673   var
674   oldPort : GrafPtr;
675   resourceOffset : integer;
676   pictureHdl : PicHandle;
677   whichPicture : integer;
678   theRect : Rect;
679
680   begin
681   resourceOffset := 0;
682
683   GetPort(oldPort);
684   SetPort(theWindowPtr);
685
686   if (activate = true) then
687     whichPicture := rToolsPict
688   else
689     whichPicture := rToolsPictDim;
690
691   if ((gColorQuickDrawPresent = false) or (gColourDisplay = false)) then
692     resourceOffset := 4;
693
694   pictureHdl := GetPicture(whichPicture + resourceOffset);
695   theRect := pictureHdl^^.picFrame;
696   OffsetRect(theRect, - theRect.left, - theRect.top);
697   if ((gColorQuickDrawPresent = false) or (gColourDisplay = false)) then
698     EraseRect(theWindowPtr^.portRect);
699   DrawPicture(pictureHdl, theRect);
700   SetWindowPic(theWindowPtr, pictureHdl);
701
702   SetPort(oldPort);
703   end;
704     {of procedure ToolsActivateHandler}
705
706 { ################################################################## ColoursActivateHandler }
707
708 procedure ColoursActivateHandler(theWindowPtr : WindowPtr; activate : boolean);
709
710   var
711   oldPort : GrafPtr;
712   resourceOffset : integer;
713   pictureHdl : PicHandle;
714   whichPicture : integer;
715   theRect : Rect;
716
717   begin
718   resourceOffset := 0;
719
720   GetPort(oldPort);
721   SetPort(theWindowPtr);
722
723   if (activate = true) then
724     whichPicture := rColoursPict
725   else
726     whichPicture := rColoursPictDim;
727
728   if ((gColorQuickDrawPresent = false) or (gColourDisplay = false)) then
729     resourceOffset := 4;
730
731   pictureHdl := GetPicture(whichPicture + resourceOffset);
732   theRect := pictureHdl^^.picFrame;
733   OffsetRect(theRect, - theRect.left, - theRect.top);
734   if ((gColorQuickDrawPresent = false) or (gColourDisplay = false)) then
735     EraseRect(theWindowPtr^.portRect);
736   DrawPicture(pictureHdl, theRect);
737   SetWindowPic(theWindowPtr, pictureHdl);
738
739   SetPort(oldPort);
740   end;
741     {of procedure ColoursActivateHandler}
742
743 { ############################################################## InvalidateScrollBarArea }
744
```

```
745  procedure InvalidateScrollBarArea(theWindowPtr : WindowPtr);
746
747    var
748    tempRect : Rect;
749
750      begin
751    SetPort(theWindowPtr);
752
753    tempRect := theWindowPtr^.portRect;
754    tempRect.left := tempRect.right - 15;
755    InvalRect(tempRect);
756
757    tempRect := theWindowPtr^.portRect;
758    tempRect.top := tempRect.bottom - 15;
759    InvalRect(tempRect);
760    end;
761      {of procedure ColoursActivateHandler}
762
763  end.
764    {of unit UFloaters.p}
765
766  { ############################################################################################ }
767
768  { ############################################################################################
769  // FloaterDemoPascal.p
770  // ############################################################################################ }
771
772  program FloatersPascal(input, output);
773
774  { ……… include the following Universal Interfaces }
775
776  uses
777
778    Windows, Fonts, Menus, TextEdit, Quickdraw, Dialogs, QuickdrawText, Processes, Types,
779    Memory, Events, TextUtils, ToolUtils, OSUtils, Devices, GestaltEqu, LowMem, SegLoad,
780
781  { ……… include the following user-defined units }
782
783    UFloaters;
784
785  { ……… global variables }
786
787  var
788
789    gColorQuickDrawPresent : boolean; external;
790    gColourDisplay : boolean; external;
791
792    theErr : OSErr;
793    response : longint;
794    mainDeviceHdl : GDHandle;
795    bitsPerPixel : integer;
796    menubarHdl : Handle;
797    menuHdl : MenuHandle;
798
799  { ################################################################ start of main program }
800
801  begin
802
803    gColorQuickDrawPresent := false;
804    gColourDisplay := false;
805
806    { ……… initialise managers }
807
808    DoInitManagers;
809
810    { ……… check for Color QuickDraw }
811
812    theErr := Gestalt(gestaltQuickdrawVersion, response);
813    if (response >= gestalt8BitQD) then
814      begin
815      gColorQuickDrawPresent := true;
816
817      mainDeviceHdl := LMGetMainDevice;
818      bitsPerPixel := mainDeviceHdl^^.gdPMap^^.pixelSize;
819      if (bitsPerPixel > 1) then
820        gColourDisplay := true;
821      end;
```

```
822
823     { ……… set up menu bar and menus }
824
825     menubarHdl := GetNewMBar(rMenubar);
826     if (menubarHdl = nil) then
827       ExitToShell;
828     SetMenuBar(menubarHdl);
829     DrawMenuBar;
830
831     menuHdl := GetMenuHandle(mApple);
832     if (menuHdl = nil) then
833       ExitToShell
834     else
835       AppendResMenu(menuHdl,'DRVR');
836
837     { ……… open floating windows }
838
839     DoOpenFloatingWindows;
840
841     CheckItem(GetMenuHandle(mFloaters), iTools, true);
842     CheckItem(GetMenuHandle(mFloaters), iColours, true);
843
844     { ……… enter EventLoop }
845
846     EventLoop;
847
848   end.
849     {of main program block}
850
851   { ########################################################################################## }
852
853   { ##########################################################################################
854   // UFloatRoutines.p                                    Routines to support floating windows
855   // ########################################################################################## }
856
857   unit UFloatRoutines;
858
859
860
861   interface
862
863   { ……… include the following Universal Interfaces }
864
865   uses
866
867     Types, Events, Quickdraw,
868
869   { ……… include the following user-defined units }
870
871     UFloaters;
872
873   { ……… exported functions and procedures }
874
875     function  NewGetNewWindow(windResourceID : integer; behind : WindowPtr;
876           activateHandler : ActivateProcPtr; windowType : integer) : WindowPtr;
877     procedure NewHideWindow(windowToHidePtr : WindowPtr);
878     procedure NewShowWindow(windowToShowPtr : WindowPtr);
879     procedure NewSelectWindow(windowToSelectPtr : WindowPtr);
880     procedure NewDragWindow(theWindowPtr : WindowPtr; startPoint : Point;
881           {$CONST} var dragBounds : Rect);
882     procedure NewDisposeWindow(theWindowPtr : WindowPtr);
883     procedure HandleSuspendEvent;
884     procedure HandleResumeEvent;
885     procedure DeactivateFloatsAndFirstDocWin;
886     procedure ActivateFloatsAndFirstDocWin;
887     function  FindFrontNonFloatingWindow : WindowPtr;
888     function  IsWindowModal(theWindowPtr : WindowPtr) : boolean;
889     function  GetWindowVisible(theWindowPtr : WindowPtr) : boolean;
890
891
892
893   implementation
894
895   { ……… include the following Universal Interfaces }
896
897   uses
898
```

```
  Windows, Fonts, Menus, TextEdit, Dialogs, QuickdrawText, Processes, LowMem,
  Memory, TextUtils, ToolUtils, OSUtils, Devices, SegLoad, GestaltEqu,

{ ……… include the following user-defined units }

  UFloaters;

{ ……… procedure and function definitions }

  procedure DeactivateWindow(theWindowPtr : WindowPtr); forward;
  procedure ActivateWindow(theWindowPtr : WindowPtr); forward;
  procedure HighlightAndActivateWindow(theWindowPtr : WindowPtr;
        activate : boolean); forward;
  function  FindLastFloatingWindow : WindowPtr; forward;
  function  GetWindowList : WindowPtr; forward;
  procedure SetWindowList(theWindowPtr : WindowPtr); forward;
  function  GetNextWindow(theWindowPtr : WindowPtr) : WindowPtr; forward;
  procedure SetNextWindow(theWindowPtr, nextWindowPtr : WindowPtr); forward;
  function  GetWasVisible(theWindowPtr : WindowPtr) : boolean; forward;
  procedure SetWasVisible(theWindowPtr : WindowPtr;  wasVisible : boolean); forward;
  function  GetWindowKind(theWindowPtr : WindowPtr) : integer; forward;
  procedure SetWindowKind(theWindowPtr : WindowPtr; windowKind : integer); forward;
  function  GetStructureRegion(theWindowPtr : WindowPtr) : RgnHandle; forward;
  function  GetContentRegion(theWindowPtr : WindowPtr) : RgnHandle; forward;
  procedure SetWindowHilite(theWindowPtr : WindowPtr; windowHilite : boolean); forward;

{ ……… procedure and function implementations }

{ ###################################################################### NewGetNewWindow }

function  NewGetNewWindow(windResourceID : integer; behind : WindowPtr;
        activateHandler : ActivateProcPtr; windowType : integer) : WindowPtr;

  var
  docRecordHdl : DocRecordHandle;
  theErr : OSErr;
  response : longint;
  newWindowPtr : WindowPtr;
  lastFloatingWindowPtr : WindowPtr;

  begin
  docRecordHdl := DocRecordHandle(NewHandle(sizeof(DocRecord)));

  theErr := Gestalt(gestaltQuickdrawVersion, response);
  if (response < gestalt32BitQD) then
    newWindowPtr := GetNewWindow(windResourceID, nil, WindowPtr(behind))
  else
    newWindowPtr := GetNewCWindow(windResourceID, nil, WindowPtr(behind));

  if (newWindowPtr <> nil) then
    begin
    SetWRefCon(newWindowPtr, longint(docRecordHdl));
    docRecordHdl := DocRecordHandle(GetWRefCon(newWindowPtr));
    docRecordHdl^^.activateHandler := activateHandler;

    if (windowType = kFloatingKind) then
      begin
      SetWindowKind(newWindowPtr, kFloatingKind);
      HiliteWindow(newWindowPtr, true);
      end
    else begin
      SetWindowKind(newWindowPtr, kDocumentKind);
      if (behind = WindowPtr(-1)) then
        begin
        lastFloatingWindowPtr := FindLastFloatingWindow;

        if (lastFloatingWindowPtr <> nil) then
          SendBehind(newWindowPtr, lastFloatingWindowPtr)
        else
          BringToFront(newWindowPtr);
        end;
      end;
    end
  else
    DisposeHandle(Handle(docRecordHdl));

  NewGetNewWindow := newWindowPtr;
```

```
  end;
    {of function NewGetNewWindow}

{ ############################################################################ NewDisposeWindow }

procedure NewDisposeWindow(theWindowPtr : WindowPtr);

  begin
  if (GetWindowVisible(theWindowPtr)) then
    NewHideWindow(theWindowPtr);
  CloseWindow(theWindowPtr);
  DisposeHandle(Handle(GetWRefCon(theWindowPtr)));
  DisposePtr(Ptr(theWindowPtr));
  end;
    {of function NewGetNewWindow}

{ ############################################################################ NewSelectWindow }

procedure NewSelectWindow(windowToSelectPtr : WindowPtr);

  var
  isFloatingWindow : boolean;
  currentFrontWindowPtr : WindowPtr;
  lastFloatingWindowPtr : WindowPtr;

  begin
  if (GetWindowKind(windowToSelectPtr) = kFloatingKind) then
    begin
    isFloatingWindow := true;
    currentFrontWindowPtr := FrontWindow;
    end
  else begin
    isFloatingWindow := false;
    currentFrontWindowPtr := FindFrontNonFloatingWindow;
    lastFloatingWindowPtr := FindLastFloatingWindow;
    end;

  if (currentFrontWindowPtr <> windowToSelectPtr) then
    begin
    if (isFloatingWindow) then
      BringToFront(windowToSelectPtr)
    else begin
      if (lastFloatingWindowPtr = nil) then
        SelectWindow(windowToSelectPtr)
      else begin
        DeactivateWindow(currentFrontWindowPtr);
        SendBehind(windowToSelectPtr, lastFloatingWindowPtr);
        ActivateWindow(windowToSelectPtr);
        end;
      end;
    end;
  end;
    {of procedure NewSelectWindow}

{ ############################################################################ NewHideWindow }

procedure NewHideWindow(windowToHidePtr : WindowPtr);

  var
  frontFloaterPtr : WindowPtr;
  frontNonFloaterPtr : WindowPtr;
  windowBehindPtr : WindowPtr;
  lastFloaterPtr : WindowPtr;

  begin
  if (GetWindowVisible(windowToHidePtr) = false) then
    Exit(NewHideWindow);

  frontFloaterPtr := FrontWindow;
  if (GetWindowKind(frontFloaterPtr) <> kFloatingKind) then
    frontFloaterPtr := nil;

  frontNonFloaterPtr := FindFrontNonFloatingWindow;

  ShowHide(windowToHidePtr, false);

  if (windowToHidePtr = frontFloaterPtr) then
```

```
      begin
      windowBehindPtr := GetNextWindow(windowToHidePtr);

      if ((windowBehindPtr <> nil) and (GetWindowKind(windowBehindPtr) = kFloatingKind)) then
        begin
        SetNextWindow(windowToHidePtr, GetNextWindow(windowBehindPtr));
        SetNextWindow(windowBehindPtr, windowToHidePtr);
        SetWindowList(windowBehindPtr);
        end;
      end
    else begin
      if (windowToHidePtr = frontNonFloaterPtr) then
        begin
        windowBehindPtr := GetNextWindow(windowToHidePtr);

        if (windowBehindPtr <> nil) then
          begin
          SetNextWindow(windowToHidePtr, GetNextWindow(windowBehindPtr));
          SetNextWindow(windowBehindPtr, windowToHidePtr);

          lastFloaterPtr := FindLastFloatingWindow;
          if (lastFloaterPtr <> nil) then
            SetNextWindow(lastFloaterPtr, windowBehindPtr)
          else
            SetWindowList(windowBehindPtr);

          ActivateWindow(windowBehindPtr);
          end;
        end;
      end;
    end;
      {of procedure NewHideWindow}

  { ############################################################################ NewShowWindow }

  procedure NewShowWindow(windowToShowPtr : WindowPtr);

    var
    windowType : integer;
    windowBehindPtr : WindowPtr;
    frontNonFloatingWindowPtr : WindowPtr;
    windowIsInFront : boolean;
    activateHandler : ActivateProcPtr;
    docRecordHdl : DocRecordHandle;

    begin
    windowIsInFront := false;

    if (GetWindowVisible(windowToShowPtr) <> false) then
      Exit(NewShowWindow);

    windowType := GetWindowKind(windowToShowPtr);

    if (windowType <> kFloatingKind) then
      begin
      windowBehindPtr := GetNextWindow(windowToShowPtr);
      if (windowBehindPtr = FindFrontNonFloatingWindow) then
        begin
        if (windowBehindPtr <> nil) then
          DeactivateWindow(windowBehindPtr);

        SetWindowHilite(windowToShowPtr, true);
        windowIsInFront := true;
        end;
      end
    else begin
      frontNonFloatingWindowPtr := FindFrontNonFloatingWindow;

      if ((frontNonFloatingWindowPtr <> nil) and
         (frontNonFloatingWindowPtr = FrontWindow) and
         (IsWindowModal(frontNonFloatingWindowPtr))) then
        begin
        SetWindowHilite(windowToShowPtr, false);
        end
      else begin
        SetWindowHilite(windowToShowPtr, true);
        windowIsInFront := true;
```

```
      end;
    end;

  ShowHide(windowToShowPtr, true);

  if (windowIsInFront) then
    begin
    docRecordHdl := DocRecordHandle(GetWRefCon(windowToShowPtr));
    activateHandler := docRecordHdl^^.activateHandler;
    activateHandler(windowToShowPtr, true);
    end;
  end;
    {of procedure NewShowWindow}

{ ############################################################################ NewDragWindow }

procedure NewDragWindow(theWindowPtr : WindowPtr; startPoint : Point;
      {$CONST} var dragBounds : Rect);

  var
  topLimit : integer;
  slopRect : Rect;
  oldPort : GrafPtr;
  windowManagerPort : GrafPtr;
  theKeyMap : KeyMap;
  commandKeyDown : boolean;
  dragRegion : RgnHandle;
  dragResult : longint;
  horizOffset : integer;
  vertOffset : integer;
  windowContentRegion : RgnHandle;
  newHorizWindowPosition : integer;
  newVertWindowPosition : integer;

  begin
  commandKeyDown := false;

  if (WaitMouseUp) then
    begin
    topLimit := GetMBarHeight + 4;
    slopRect := dragBounds;

    if (slopRect.top < topLimit) then
      slopRect.top := topLimit;

    GetPort(oldPort);
    GetWMgrPort(windowManagerPort);
    SetPort(windowManagerPort);

    SetClip(GetGrayRgn);

    GetKeys(theKeyMap);
    if (BAnd(theKeyMap[1], $8000) <> 0) then
      commandKeyDown := true;

    if ((commandKeyDown = true) or
      (GetWindowKind(theWindowPtr) <> kFloatingKind)) then
      begin
      if (commandKeyDown = false) then
        ClipAbove(WindowRef(WindowPeek(FindFrontNonFloatingWindow)))
      else
        ClipAbove(WindowRef(WindowPeek(theWindowPtr)));
      end;

    dragRegion := NewRgn;
    CopyRgn(GetStructureRegion(theWindowPtr), dragRegion);

    dragResult := DragGrayRgn(dragRegion, startPoint, slopRect, slopRect,
          noConstraint, nil);

    SetPort(oldPort);

    if (dragResult <> 0) then
      begin
      horizOffset := BAnd(dragResult, $FFFF);
      vertOffset := BSR(dragResult, 16);
```

```
1207        if (vertOffset <> -32768) then
1208          begin
1209          windowContentRegion := GetContentRegion(theWindowPtr);
1210
1211          newHorizWindowPosition := windowContentRegion^^.rgnBBox.left
1212                                    + horizOffset;
1213          newVertWindowPosition := windowContentRegion^^.rgnBBox.top + vertOffset;
1214
1215          MoveWindow(theWindowPtr, newHorizWindowPosition, newVertWindowPosition,
1216                  false);
1217          end;
1218        end;
1219
1220      if (commandKeyDown = false) then
1221        NewSelectWindow(theWindowPtr);
1222
1223      DisposeRgn(dragRegion);
1224      end;
1225    end;
1226      {of procedure NewDragWindow}
1227
1228  { ###################################################################### HandleSuspendEvent }
1229
1230  procedure HandleSuspendEvent;
1231
1232    var
1233    currentWindowPtr : WindowPtr;
1234    windowIsVisible : boolean;
1235
1236    begin
1237    currentWindowPtr := GetWindowList;
1238
1239    if (GetWindowKind(currentWindowPtr) <> kFloatingKind) then
1240      Exit(HandleSuspendEvent);
1241
1242    while ((currentWindowPtr <> nil) &
1243            (GetWindowKind(currentWindowPtr) = kFloatingKind)) do
1244      begin
1245      windowIsVisible := GetWindowVisible(currentWindowPtr);
1246      SetWasVisible(currentWindowPtr, windowIsVisible);
1247      if (windowIsVisible) then
1248        ShowHide(currentWindowPtr, false);
1249      currentWindowPtr := GetNextWindow(currentWindowPtr);
1250      end;
1251
1252    currentWindowPtr := FindFrontNonFloatingWindow;
1253    if (currentWindowPtr <> nil) then
1254      begin
1255      DrawGrowIcon(currentWindowPtr);
1256      DeactivateWindow(currentWindowPtr);
1257      end;
1258    end;
1259      {of procedure HandleSuspendEvent}
1260
1261  { ###################################################################### HandleResumeEvent }
1262
1263  procedure HandleResumeEvent;
1264
1265    var
1266    currentWindowPtr : WindowPtr;
1267    windowWasVisible : boolean;
1268
1269    begin
1270    currentWindowPtr := GetWindowList;
1271
1272    if (GetWindowKind(currentWindowPtr) <> kFloatingKind) then
1273      Exit(HandleResumeEvent);
1274
1275    while((currentWindowPtr <> nil) &
1276            (GetWindowKind(currentWindowPtr) = kFloatingKind)) do
1277      begin
1278      windowWasVisible := GetWasVisible(currentWindowPtr);
1279      if (windowWasVisible) then
1280        begin
1281        ShowHide(currentWindowPtr, true);
1282        ActivateWindow(currentWindowPtr);
1283        end;
```

```
1284      currentWindowPtr := GetNextWindow(currentWindowPtr);
1285      end;
1286
1287    currentWindowPtr := FindFrontNonFloatingWindow;
1288    if (currentWindowPtr <> nil) then
1289      begin
1290      DrawGrowIcon(currentWindowPtr);
1291      ActivateWindow(currentWindowPtr);
1292      end;
1293    end;
1294      {of procedure HandleResumeEvent}
1295
1296  { ###################################################### DeactivateFloatsAndFirstDocWin }
1297
1298  procedure DeactivateFloatsAndFirstDocWin;
1299
1300    var
1301    firstWindowPtr : WindowPtr;
1302    secondDocumentWindowPtr : WindowPtr;
1303    currentWindowPtr : WindowPtr;
1304
1305    begin
1306    firstWindowPtr := FrontWindow;
1307    secondDocumentWindowPtr := FindFrontNonFloatingWindow;
1308    if (secondDocumentWindowPtr <> nil) then
1309      secondDocumentWindowPtr := GetNextWindow(secondDocumentWindowPtr);
1310
1311    currentWindowPtr := firstWindowPtr;
1312    while (currentWindowPtr <> secondDocumentWindowPtr) do
1313      begin
1314      if (GetWindowVisible(currentWindowPtr)) then
1315        DeactivateWindow(currentWindowPtr);
1316      currentWindowPtr := GetNextWindow(currentWindowPtr);
1317      end;
1318    end;
1319      {of procedure DeactivateFloatsAndFirstDocWin}
1320
1321  { ###################################################### ActivateFloatsAndFirstDocWin }
1322
1323  procedure ActivateFloatsAndFirstDocWin;
1324
1325    var
1326    getFrontProcessResult : OSErr;
1327    getCurrentProcessResult : OSErr;
1328    frontPSN : ProcessSerialNumber;
1329    currentPSN : ProcessSerialNumber;
1330    sameProcessResult : OSErr;
1331    isSameProcess : boolean;
1332    firstWindowPtr : WindowPtr;
1333    secondDocumentWindowPtr : WindowPtr;
1334    currentWindowPtr : WindowPtr;
1335
1336    begin
1337    getFrontProcessResult := GetFrontProcess(frontPSN);
1338    getCurrentProcessResult := GetCurrentProcess(currentPSN);
1339
1340    if ((getFrontProcessResult = noErr) and (getCurrentProcessResult = noErr)) then
1341      sameProcessResult := SameProcess(frontPSN, currentPSN, isSameProcess);
1342
1343    if ((sameProcessResult = noErr) and (isSameProcess = false)) then
1344      HandleSuspendEvent
1345    else begin
1346      firstWindowPtr := FrontWindow;
1347      secondDocumentWindowPtr := FindFrontNonFloatingWindow;
1348      if (secondDocumentWindowPtr <> nil) then
1349        secondDocumentWindowPtr := GetNextWindow(secondDocumentWindowPtr);
1350      currentWindowPtr := firstWindowPtr;
1351
1352      while(currentWindowPtr <> secondDocumentWindowPtr) do
1353        begin
1354        if (GetWindowVisible(currentWindowPtr)) then
1355          ActivateWindow(currentWindowPtr);
1356        currentWindowPtr := GetNextWindow(currentWindowPtr);
1357        end;
1358      end;
1359    end;
1360      {of procedure ActivateFloatsAndFirstDocWin}
```

```
{ ############################################################################ DeactivateWindow }

procedure DeactivateWindow(theWindowPtr : WindowPtr);

  begin
  HighlightAndActivateWindow(theWindowPtr, false);
  end;
    {of procedure DeactivateWindow}

{ ############################################################################## ActivateWindow }

procedure ActivateWindow(theWindowPtr : WindowPtr);

  begin
  HighlightAndActivateWindow(theWindowPtr, true);
  end;
    {of procedure ActivateWindow}

{ ########################################################### HighlightAndActivateWindow }

procedure HighlightAndActivateWindow(theWindowPtr : WindowPtr;  activate : boolean);

  var
  activateHandler : ActivateProcPtr;
  docRecordHdl : DocRecordHandle;

  begin
  HiliteWindow(theWindowPtr, activate);
  docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));
  activateHandler := docRecordHdl^^.activateHandler;
  if (activateHandler <> nil) then
    activateHandler(theWindowPtr, activate);
  end;
    {of procedure HighlightAndActivateWindow}

{ ############################################################ FindFrontNonFloatingWindow }

function  FindFrontNonFloatingWindow : WindowPtr;

  var
  frontWindowPtr : WindowPtr;

  begin
  frontWindowPtr := FrontWindow;

  while (frontWindowPtr <> nil) & ((GetWindowKind(frontWindowPtr) = kFloatingKind) or
      not GetWindowVisible(frontWindowPtr)) do
    frontWindowPtr := GetNextWindow(frontWindowPtr);

  FindFrontNonFloatingWindow := frontWindowPtr;
  end;
    {of function FindFrontNonFloatingWindow}

{ ############################################################## FindLastFloatingWindow }

function  FindLastFloatingWindow : WindowPtr;

  var
  theWindowPtr : WindowPtr;
  lastFloatingWindowPtr : WindowPtr;

  begin
  theWindowPtr := GetWindowList;
  lastFloatingWindowPtr := nil;

  while (theWindowPtr <> nil) do
    begin
    if (GetWindowKind(theWindowPtr) = kFloatingKind) then
      lastFloatingWindowPtr := theWindowPtr;
    theWindowPtr := GetNextWindow(theWindowPtr);
    end;

  FindLastFloatingWindow := lastFloatingWindowPtr;
  end;
    {of function FindLastFloatingWindow}
```

```
{ ############################################################################ IsWindowModal }

function  IsWindowModal(theWindowPtr : WindowPtr) : boolean;

  var
  variant : integer;

  begin
  variant := GetWVariant(theWindowPtr);

  if (theWindowPtr = nil) then
    IsWindowModal := false
  else if ((WindowPeek(theWindowPtr)^.windowKind = dialogKind) and
     ((variant = dBoxProc) or (variant = movableDBoxProc))) then
    IsWindowModal := true
  else
    IsWindowModal := false;
  end;
    {of function IsWindowModal}

{ ############################################################################ GetWindowList }

function  GetWindowList : WindowPtr;

  begin
  GetWindowList := LMGetWindowList;
  end;
    {of function GetWindowList}

{ ############################################################################ SetWindowList }

procedure SetWindowList(theWindowPtr : WindowPtr);

  begin
  LMSetWindowList(theWindowPtr);
  end;
    {of procedure SetWindowList}


{ ############################################################################ GetNextWindow }

function  GetNextWindow(theWindowPtr : WindowPtr) : WindowPtr;

  begin
  GetNextWindow := WindowPtr(WindowPeek(theWindowPtr)^.nextWindow);
  end;
    {of function GetNextWindow}


{ ############################################################################ SetNextWindow }

procedure SetNextWindow(theWindowPtr, nextWindowPtr : WindowPtr);

  begin
  WindowPeek(theWindowPtr)^.nextWindow := WindowPeek(nextWindowPtr);
  end;
    {of procedure SetNextWindow}


{ ############################################################################ GetWasVisible }

function  GetWasVisible(theWindowPtr : WindowPtr) : boolean;

  var
  docRecordHdl : DocRecordHandle;

  begin
  docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));
  GetWasVisible := docRecordHdl^^.wasVisible;
  end;
    {of function GetWasVisible}


{ ############################################################################ SetWasVisible }

procedure SetWasVisible(theWindowPtr : WindowPtr;  wasVisible : boolean);

```

```
  var
  docRecordHdl : DocRecordHandle;

  begin
  docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));
  docRecordHdl^^.wasVisible := wasVisible;
  end;
    {of procedure SetWasVisible}


{ ############################################################################ GetWindowKind }

function  GetWindowKind(theWindowPtr : WindowPtr) : integer;

  var
  docRecordHdl : DocRecordHandle;

  begin
  if (WindowPeek(theWindowPtr)^.windowKind <> dialogKind) then
    begin
    docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));
    if (docRecordHdl <> nil) then
      GetWindowKind := docRecordHdl^^.windowType;
    end
  else
    GetWindowKind := 0;
  end;
    {of function GetWindowKind}


{ ############################################################################ SetWindowKind }

procedure SetWindowKind(theWindowPtr : WindowPtr; windowKind : integer);

  var
  docRecordHdl : DocRecordHandle;

  begin
  docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));
  docRecordHdl^^.windowType := windowKind;
  end;
    {of procedure SetWindowKind}

{ ########################################################################## GetWindowVisible }

function  GetWindowVisible(theWindowPtr : WindowPtr) : boolean;

  begin
  GetWindowVisible := WindowPeek(theWindowPtr)^.visible;
  end;
    {of function GetWindowVisible}


{ ######################################################################## GetStructureRegion }

function  GetStructureRegion(theWindowPtr : WindowPtr) : RgnHandle;

  begin
  GetStructureRegion := WindowPeek(theWindowPtr)^.strucRgn;
  end;
    {of function GetStructureRegion}


{ ########################################################################## GetContentRegion }

function  GetContentRegion(theWindowPtr : WindowPtr) : RgnHandle;

  begin
  GetContentRegion := WindowPeek(theWindowPtr)^.contRgn;
  end;
    {of function GetContentRegion}


{ ############################################################################ SetWindowHilite }

procedure SetWindowHilite(theWindowPtr : WindowPtr; windowHilite : boolean);

```

```
  begin
  WindowPeek(theWindowPtr)^.hilited := windowHilite;
  end;
    {of procedure SetWindowHilite}

end.
  {of unit UFloatRoutines.p}

{ ############################################################################################ }

{ ############################################################################################
// WDEFPascal.p                    Custom Window Definition Function for Floating Windows
// ############################################################################################
//
// This WDEF creates a utility window whose appearance conforms to that specified in the
// document titled Apple Grayscale Appearance for System 7.5 and published by Apple
// Computer, Inc. On black-and-white displays, the WDEF is drawn in black-and-white with
// an appearance similar to the black-and-white floating windows found in many commercial
// applications. The WDEF supports only one variation code. It provides for a close box
// but not for a zoom box or window title.
//
// The WDEF utilises three 'cicn' resources (nonpurgeable), one for the close box in the
// normal state, one for the close box in the pressed state, and one to paint the
// checkered pattern in the title bar.
//
// ############################################################################################ }

unit WDEFPascal;

{ ……… unit interface section }

interface

{ ……… include the following Universal Interfaces }

uses

  Types, Windows, Icons;

{ ……… define the following constants }

const

rCloseEnabledIcon = 128;
rClosePressedIcon = 129;
rCheckPatternIcon = 130;

{ ……… main procedure interface }

{$MAIN}
  function main(varCode : integer; theWindowPeek : WindowPeek; message : integer;
        param : longint) : longint;



{ ……… unit implementation section }

implementation

{ ……… include the following Universal Interfaces }

uses

  Windows, Fonts, Menus, Quickdraw, ToolUtils, OSUtils, Devices, LowMem,
  PascalA4, GestaltEqu;

{ ……… user-defined types }

type

Globals = record
  gColourQuickDrawPresent : boolean;
  gColourDisplay : boolean;
  gBlackPattern : Pattern;
  gCloseEnabledHdl : CIconHandle;
  gClosePressedHdl : CIconHandle;
  gCheckPatternHdl : CIconHandle;
```

```
    gWhite : RGBColor;
    gGray1 : RGBColor;
    gGray2 : RGBColor;
    gGray3 : RGBColor;
    gGray4 : RGBColor;
    gGray6 : RGBColor;
    gGray7 : RGBColor;
    gGray8 : RGBColor;
    gGray10 : RGBColor;
    gBlack : RGBColor;
    gToggle : boolean;
    end;
  GlobalsPointer = ^Globals;
  GlobalsHandle = ^GlobalsPointer;

  { ……procedure and function interfaces }

    procedure DoInitMessage(theWindowPeek : WindowPeek); forward;
    procedure DoDrawMessage(theWindowPeek : WindowPeek; param : longint); forward;
    function  DoHitMessage(theWindowPeek : WindowPeek; param : longint) : longint; forward;
    procedure DoCalcRgnsMessage(theWindowPeek : WindowPeek); forward;
    procedure DrawWindowColour(theWindowPeek : WindowPeek); forward;
    procedure DrawWindowMono(theWindowPeek : WindowPeek); forward;
    procedure ToggleGoAway(theWindowPeek : WindowPeek); forward;
    procedure DrawGoAwayBox(theWindowPeek : WindowPeek); forward;
    procedure DrawGoAwayBoxPressed(theWindowPeek : WindowPeek); forward;
    procedure GetGoAwayRect(theWindowPeek : WindowPeek; var theRect : Rect); forward;
    procedure GetContentRect(theWindowPeek : WindowPeek; var theRect : Rect); forward;
    procedure GetStructRect(theWindowPeek : WindowPeek; var theRect : Rect); forward;
    procedure SyncPorts; forward;

  { ############################################################################## main }

  function  main(varCode : integer; theWindowPeek : WindowPeek; message : integer;
        param : longint) : longint;

    var
    result : longint;
    oldPort : GrafPtr;
    oldPenState : PenState;
    oldA4, ignoredLong : longint;

    begin
    oldA4 := SetCurrentA4;

    GetPenState(oldPenState);
    GetPort(oldPort);

    if (theWindowPeek^.dataHandle <> nil) then
      if (GlobalsHandle(theWindowPeek^.dataHandle)^^.gColourQuickDrawPresent) then
        SyncPorts;

    result := 0;

    case (message) of

      wNew: begin
        DoInitMessage(theWindowPeek);
        end;

      wDraw: begin
        if (theWindowPeek^.visible) then
          DoDrawMessage(theWindowPeek, param);
        end;

      wHit: begin
        result := DoHitMessage(theWindowPeek, param);
        end;

      wCalcRgns: begin
        DoCalcRgnsMessage(theWindowPeek);
        end;
      end;
        {of case statement}

    SetPenState(oldPenState);
    SetPort(oldPort);
```

```
1746    main := result;
1747
1748    ignoredLong := SetA4(oldA4);
1749    end;
1750      {of function main}
1751
1752  { ############################################################################ DoInitMessage }
1753
1754  procedure DoInitMessage(theWindowPeek : WindowPeek);
1755
1756    var
1757    theErr : OSErr;
1758    response : longint;
1759    mainDeviceHdl : GDHandle;
1760    bitsPerPixel, a : integer;
1761    dataHdl : GlobalsHandle;
1762
1763    begin
1764    dataHdl := GlobalsHandle(NewHandleClear(sizeof(Globals)));
1765
1766    if (dataHdl <> nil) then
1767      begin
1768      dataHdl^^.gWhite.red := $FFFF;
1769      dataHdl^^.gWhite.blue := $FFFF;
1770      dataHdl^^.gWhite.green := $FFFF;
1771      dataHdl^^.gGray1.red := $EEEE;
1772      dataHdl^^.gGray1.blue := $EEEE;
1773      dataHdl^^.gGray1.green := $EEEE;
1774      dataHdl^^.gGray2.red := $DDDD;
1775      dataHdl^^.gGray2.blue := $DDDD;
1776      dataHdl^^.gGray2.green := $DDDD;
1777      dataHdl^^.gGray3.red := $CCCC;
1778      dataHdl^^.gGray3.blue := $CCCC;
1779      dataHdl^^.gGray3.green := $CCCC;
1780      dataHdl^^.gGray4.red := $BBBB;
1781      dataHdl^^.gGray4.blue := $BBBB;
1782      dataHdl^^.gGray4.green := $BBBB;
1783      dataHdl^^.gGray6.red := $9999;
1784      dataHdl^^.gGray6.blue := $9999;
1785      dataHdl^^.gGray6.green := $9999;
1786      dataHdl^^.gGray7.red := $8888;
1787      dataHdl^^.gGray7.blue := $8888;
1788      dataHdl^^.gGray7.green := $8888;
1789      dataHdl^^.gGray8.red := $7777;
1790      dataHdl^^.gGray8.blue := $7777;
1791      dataHdl^^.gGray8.green := $7777;
1792      dataHdl^^.gGray10.red := $5555;
1793      dataHdl^^.gGray10.blue := $5555;
1794      dataHdl^^.gGray10.green := $5555;
1795      dataHdl^^.gBlack.red := $0000;
1796      dataHdl^^.gBlack.blue := $0000;
1797      dataHdl^^.gBlack.green := $0000;
1798
1799      dataHdl^^.gColourQuickDrawPresent := false;
1800      dataHdl^^.gColourDisplay := false;
1801      dataHdl^^.gCloseEnabledHdl := nil;
1802      dataHdl^^.gClosePressedHdl := nil;
1803      dataHdl^^.gCheckPatternHdl := nil;
1804
1805      theErr := Gestalt(gestaltQuickdrawVersion, response);
1806      if (response >= gestalt8BitQD) then
1807        begin
1808        dataHdl^^.gColourQuickDrawPresent := true;
1809
1810        mainDeviceHdl := LMGetMainDevice;
1811        bitsPerPixel := mainDeviceHdl^^.gdPMap^^.pixelSize;
1812        if (bitsPerPixel > 1) then
1813          dataHdl^^.gColourDisplay := true;
1814        end;
1815
1816      for a := 0 to 7 do
1817        dataHdl^^.gBlackPattern.pat[a] := -1;
1818
1819      if (dataHdl^^.gColourQuickDrawPresent) then
1820        begin
1821        dataHdl^^.gCloseEnabledHdl := GetCIcon(rCloseEnabledIcon);
1822        dataHdl^^.gClosePressedHdl := GetCIcon(rClosePressedIcon);
```

```
      dataHdl^^.gCheckPatternHdl := GetCIcon(rCheckPatternIcon);
      end;

    dataHdl^^.gToggle := false;

    theWindowPeek^.dataHandle := Handle(dataHdl);
    end;
  end;
    {of procedure DoInitMessage}

{ ############################################################################ DoDrawMessage }

procedure DoDrawMessage(theWindowPeek : WindowPeek; param : longint);

  var
  tempLong : longint;
  dataHdl : GlobalsHandle;

  begin
  dataHdl := GlobalsHandle(theWindowPeek^.dataHandle);
  tempLong := BAnd(param, $0000FFFF);
  param := tempLong;

  case (param) of

    wNoHit: begin
      if (dataHdl^^.gColourQuickDrawPresent and dataHdl^^.gColourDisplay) then
        DrawWindowColour(theWindowPeek)
      else
        DrawWindowMono(theWindowPeek);
      end;

    wInGoAway: begin
      ToggleGoAway(theWindowPeek);
      end;

    otherwise begin
      end;

    end;
      {of case statement}
  end;
    {of procedure DoDrawMessage}

{ ############################################################################ DoHitMessage }

function  DoHitMessage(theWindowPeek : WindowPeek; param : longint) : longint;

  var
  where : Point;
  goAwayRect : Rect;

  begin
  where.v := HiWord(param);
  where.h := LoWord(param);

  if (PtInRgn(where, theWindowPeek^.contRgn)) then
    begin
    DoHitMessage := wInContent;
    Exit(DoHitMessage);
    end
  else if (PtInRgn(where, theWindowPeek^.strucRgn)) then
    begin
    if (theWindowPeek^.goAwayFlag) then
      begin
      GetGoAwayRect(theWindowPeek, goAwayRect);
      if (PtInRect(where, goAwayRect)) then
        begin
        DoHitMessage := wInGoAway;
        Exit(DoHitMessage);
        end;
      end;

    DoHitMessage := wInDrag;
    Exit(DoHitMessage);
    end;

```

```
1900    DoHitMessage := wNoHit;
1901    end;
1902      {of procedure DoHitMessage}
1903
1904  { ###################################################################### DoCalcRgnsMessage }
1905
1906  procedure DoCalcRgnsMessage(theWindowPeek : WindowPeek);
1907
1908    var
1909    tempRgn : RgnHandle;
1910    theRect : Rect;
1911
1912    begin
1913    tempRgn := NewRgn;
1914
1915    GetContentRect(theWindowPeek, theRect);
1916    RectRgn(theWindowPeek^.contRgn, theRect);
1917
1918    GetStructRect(theWindowPeek, theRect);
1919    RectRgn(theWindowPeek^.strucRgn, theRect);
1920    OffsetRect(theRect, 1, 1);
1921    theRect.left := theRect.left + 1;
1922    theRect.top := theRect.top + 1;
1923    RectRgn(tempRgn, theRect);
1924    UnionRgn(tempRgn, theWindowPeek^.strucRgn, theWindowPeek^.strucRgn);
1925
1926    DisposeRgn(tempRgn);
1927    end;
1928      {of procedure DoCalcRgnsMessage}
1929
1930  { ###################################################################### DrawWindowColour }
1931
1932  procedure DrawWindowColour(theWindowPeek : WindowPeek);
1933
1934    var
1935    oldForeColour : RGBColor;
1936    oldBackColour : RGBColor;
1937    contentRect, structRect, theRect : Rect;
1938    a, b : integer;
1939    dataHdl : GlobalsHandle;
1940
1941    begin
1942    dataHdl := GlobalsHandle(theWindowPeek^.dataHandle);
1943
1944    GetForeColor(oldForeColour);
1945    GetBackColor(oldBackColour);
1946    PenSize(1, 1);
1947    PenPat(dataHdl^^.gBlackPattern);
1948    PenMode(patCopy);
1949
1950    if (theWindowPeek^.hilited) then
1951      begin
1952      GetContentRect(theWindowPeek, contentRect);
1953
1954      RGBForeColor(dataHdl^^.gBlack);
1955      InsetRect(contentRect, -1, -1);
1956      FrameRect(contentRect);
1957
1958      GetStructRect(theWindowPeek, structRect);
1959
1960      RGBForeColor(dataHdl^^.gGray3);
1961      SetRect(theRect, structRect.left + 2, structRect.top + 2, structRect.right - 2,
1962              structRect.top + 12);
1963      PaintRect(theRect);
1964
1965      SetRect(theRect, structRect.left + 14, structRect.top + 3, structRect.left + 22,
1966            structRect.top + 11);
1967      b := (((structRect.right - 4) - (structRect.left + 14)) div 9) + 1;
1968      a := 0;
1969      while a < b + 1 do
1970        begin
1971        PlotCIcon(theRect, dataHdl^^.gCheckPatternHdl);
1972        OffsetRect(theRect, 9, 0);
1973        a := a + 1;
1974        end;
1975
1976      if (theWindowPeek^.goAwayFlag) then
```

```
1977        DrawGoAwayBox(theWindowPeek);
1978
1979      RGBForeColor(dataHdl^^.gBlack);
1980      FrameRect(structRect);
1981      MoveTo(structRect.left + 2, structRect.bottom);
1982      LineTo(structRect.right, structRect.bottom);
1983      LineTo(structRect.right, structRect.top + 2);
1984
1985      RGBForeColor(dataHdl^^.gWhite);
1986      MoveTo(structRect.left + 1, structRect.bottom - 3);
1987      LineTo(structRect.left + 1, structRect.top + 1);
1988      LineTo(structRect.right - 3, structRect.top + 1);
1989      MoveTo(structRect.left + 3, structRect.top + 11);
1990      LineTo(structRect.left + 11, structRect.top + 11);
1991      LineTo(structRect.left + 11, structRect.top + 3);
1992
1993      RGBForeColor(dataHdl^^.gGray1);
1994      MoveTo(structRect.left + 2, structRect.bottom - 3);
1995      LineTo(structRect.right - 3, structRect.bottom - 3);
1996      LineTo(structRect.right - 3, structRect.top + 13);
1997
1998      RGBForeColor(dataHdl^^.gGray3);
1999      MoveTo(structRect.right - 2, structRect.top + 1);
2000      LineTo(structRect.right - 2, structRect.top + 1);
2001      RGBForeColor(dataHdl^^.gGray4);
2002      MoveTo(structRect.left + 1, structRect.bottom - 2);
2003      LineTo(structRect.left + 1, structRect.bottom - 2);
2004
2005      RGBForeColor(dataHdl^^.gGray6);
2006      MoveTo(structRect.left + 2, structRect.bottom - 2);
2007      LineTo(structRect.right - 2, structRect.bottom - 2);
2008      LineTo(structRect.right - 2, structRect.top + 2);
2009      MoveTo(structRect.right - 3, structRect.top + 12);
2010      LineTo(structRect.left + 2, structRect.top + 12);
2011      LineTo(structRect.left + 2, structRect.bottom - 4);
2012
2013      RGBForeColor(dataHdl^^.gGray7);
2014      MoveTo(structRect.left + 2, structRect.top + 10);
2015      LineTo(structRect.left + 2, structRect.top + 2);
2016      LineTo(structRect.left + 10, structRect.top + 2);
2017      end
2018    else begin
2019      RGBForeColor(dataHdl^^.gGray10);
2020      GetContentRect(theWindowPeek, contentRect);
2021      InsetRect(contentRect, -1, -1);
2022      FrameRect(contentRect);
2023
2024      GetStructRect(theWindowPeek, structRect);
2025      FrameRect(structRect);
2026      MoveTo(structRect.left + 2, structRect.bottom);
2027      LineTo(structRect.right, structRect.bottom);
2028      LineTo(structRect.right, structRect.top + 2);
2029
2030      RGBForeColor(dataHdl^^.gGray2);
2031      InsetRect(structRect, 1, 1);
2032      PenSize(2, 2);
2033      FrameRect(structRect);
2034      structRect.bottom := structRect.top + 12;
2035      PaintRect(structRect);
2036      end;
2037
2038    RGBForeColor(oldForeColour);
2039    RGBBackColor(oldBackColour);
2040    end;
2041      {of procedure DrawWindowColour}
2042
2043  { ############################################################################ DrawWindowMono }
2044
2045  procedure DrawWindowMono(theWindowPeek : WindowPeek);
2046
2047    var
2048    contentRect, structRect, theRect : Rect;
2049    a, b : integer;
2050    pattern : UInt8;
2051    checkPattern : Pattern;
2052    dataHdl : GlobalsHandle;
2053
```

```
2054    begin
2055    dataHdl := GlobalsHandle(theWindowPeek^.dataHandle);
2056
2057    PenSize(1, 1);
2058    PenPat(dataHdl^^.gBlackPattern);
2059    PenMode(patCopy);
2060
2061    if (theWindowPeek^.goAwayFlag and theWindowPeek^.hilited) then
2062      begin
2063      ForeColor(blackColor);
2064      BackColor(whiteColor);
2065
2066      GetContentRect(theWindowPeek, contentRect);
2067
2068      InsetRect(contentRect, -1, -1);
2069      FrameRect(contentRect);
2070
2071      GetStructRect(theWindowPeek, structRect);
2072
2073      FrameRect(structRect);
2074      MoveTo(structRect.left + 2, structRect.bottom);
2075      LineTo(structRect.right, structRect.bottom);
2076      LineTo(structRect.right, structRect.top + 2);
2077
2078      SetRect(theRect, structRect.left + 1, structRect.top + 1, structRect.right - 1,
2079              structRect.top + 10);
2080      EraseRect(theRect);
2081
2082      for a := 0 to 7 do
2083        checkPattern.pat[a] := $00;
2084
2085      if (BAnd(structRect.left, 1) <> 0) then
2086        pattern := $AA
2087      else
2088        pattern := $55;
2089
2090      if (BAnd(structRect.top, 1) <> 0) then
2091        b := 1
2092      else
2093        b := 0;
2094
2095      a := b;
2096
2097      while(a < 8) do
2098        begin
2099        checkPattern.pat[a] := pattern;
2100        a := a + 2;
2101        end;
2102
2103      PenPat(checkPattern);
2104      SetRect(theRect, structRect.left + 11, structRect.top + 2, structRect.right - 2,
2105              structRect.top + 9);
2106      PaintRect(theRect);
2107
2108      PenPat(dataHdl^^.gBlackPattern);
2109
2110      if (theWindowPeek^.goAwayFlag) then
2111        DrawGoAwayBox(theWindowPeek);
2112      end
2113    else begin
2114      SetRect(theRect, structRect.left + 1, structRect.top + 1, structRect.right - 1,
2115              structRect.top + 13);
2116      EraseRect(theRect);
2117      end;
2118    end;
2119      {of procedure DrawWindowMono}
2120
2121  { ############################################################################ ToggleGoAway }
2122
2123  procedure ToggleGoAway(theWindowPeek : WindowPeek);
2124
2125    var
2126    dataHdl : GlobalsHandle;
2127
2128    begin
2129    dataHdl := GlobalsHandle(theWindowPeek^.dataHandle);
2130
```

```
2131    dataHdl^^.gToggle := not (dataHdl^^.gToggle);
2132
2133    if (dataHdl^^.gToggle) then
2134      DrawGoAwayBoxPressed(theWindowPeek)
2135    else
2136      DrawGoAwayBox(theWindowPeek);
2137    end;
2138      {of procedure ToggleGoAway}
2139
2140  { ############################################################################ DrawGoAwayBox }
2141
2142  procedure DrawGoAwayBox(theWindowPeek : WindowPeek);
2143
2144    var
2145    theRect : Rect;
2146    dataHdl : GlobalsHandle;
2147
2148    begin
2149    dataHdl := GlobalsHandle(theWindowPeek^.dataHandle);
2150
2151    GetGoAwayRect(theWindowPeek, theRect);
2152
2153    if (dataHdl^^.gColourQuickDrawPresent and dataHdl^^.gColourDisplay) then
2154      PlotCIcon(theRect, dataHdl^^.gCloseEnabledHdl)
2155    else begin
2156      EraseRect(theRect);
2157      PenSize(1, 1);
2158      FrameRect(theRect);
2159      end;
2160    end;
2161      {of procedure DrawGoAwayBox}
2162
2163  { ######################################################################## DrawGoAwayBoxPressed }
2164
2165  procedure DrawGoAwayBoxPressed(theWindowPeek : WindowPeek);
2166
2167    var
2168    theRect : Rect;
2169    dataHdl : GlobalsHandle;
2170
2171    begin
2172    dataHdl := GlobalsHandle(theWindowPeek^.dataHandle);
2173
2174    GetGoAwayRect(theWindowPeek, theRect);
2175
2176    if (dataHdl^^.gColourQuickDrawPresent and dataHdl^^.gColourDisplay) then
2177      PlotCIcon(theRect, dataHdl^^.gClosePressedHdl)
2178    else begin
2179      PenSize(2, 2);
2180      FrameRect(theRect);
2181      end;
2182    end;
2183      {of procedure DrawGoAwayBox}
2184
2185  { ############################################################################ GetGoAwayRect }
2186
2187  procedure GetGoAwayRect(theWindowPeek : WindowPeek; var theRect : Rect);
2188
2189    var
2190    dataHdl : GlobalsHandle;
2191
2192    begin
2193    dataHdl := GlobalsHandle(theWindowPeek^.dataHandle);
2194
2195    GetStructRect(theWindowPeek, theRect);
2196
2197    if (dataHdl^^.gColourQuickDrawPresent and dataHdl^^.gColourDisplay) then
2198      begin
2199      theRect.top := theRect.top + 3;
2200      theRect.left := theRect.left + 3;
2201      theRect.bottom := theRect.top + 8;
2202      theRect.right := theRect.left + 8;
2203      end
2204    else begin
2205      theRect.top := theRect.top + 2;
2206      theRect.left := theRect.left + 2;
2207      theRect.bottom := theRect.top + 7;
```

```
2208      theRect.right := theRect.left + 7;
2209      end;
2210    end;
2211      {of procedure GetGoAwayRect}
2212
2213  { ############################################################################ GetContentRect }
2214
2215  procedure GetContentRect(theWindowPeek : WindowPeek; var theRect : Rect);
2216
2217    var
2218    oldPort : GrafPtr;
2219
2220    begin
2221    theRect := theWindowPeek^.port.portRect;
2222
2223    GetPort(oldPort);
2224    SetPort(GrafPtr(theWindowPeek));
2225
2226    LocalToGlobal(theRect.topLeft);
2227    LocalToGlobal(theRect.botRight);
2228
2229    SetPort(oldPort);
2230    end;
2231      {of procedure GetContentRect}
2232
2233  { ############################################################################ GetStructRect }
2234
2235  procedure GetStructRect(theWindowPeek : WindowPeek; var theRect : Rect);
2236
2237    var
2238    dataHdl : GlobalsHandle;
2239
2240    begin
2241    dataHdl := GlobalsHandle(theWindowPeek^.dataHandle);
2242
2243    GetContentRect(theWindowPeek, theRect);
2244
2245    if (dataHdl^^.gColourQuickDrawPresent and dataHdl^^.gColourDisplay) then
2246      begin
2247      InsetRect(theRect, -4, -4);
2248      theRect.top := theRect.top - 10;
2249      end
2250    else begin
2251      theRect.top := theRect.top - 10;
2252      InsetRect(theRect, -1, -1);
2253      end;
2254    end;
2255      {of procedure GetStructRect}
2256
2257  { ################################################################################ SyncPorts }
2258
2259  procedure SyncPorts;
2260
2261    var
2262    bwPort : GrafPtr;
2263    colourPort : CGrafPtr;
2264
2265    begin
2266    GetWMgrPort(bwPort);
2267    GetCWMgrPort(colourPort);
2268    SetPort(GrafPtr(colourPort));
2269
2270    BlockMoveData(@bwPort^.pnLoc, @colourPort^.pnLoc, 10);
2271    BlockMoveData(@bwPort^.pnVis, @colourPort^.pnVis, 14);
2272
2273    PenPat(bwPort^.pnPat);
2274    BackPat(bwPort^.bkPat);
2275    end;
2276      {of procedure SyncPorts}
2277
2278  end.
2279    {of unit WDEFPascal}
2280
2281  { ########################################################################################## }
```

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.

UFloaters.p

Because the source code is divided into three files (FloaterDemoPascal.p, UFloaters.p and UFloatRoutines.p), most constants, type definitions and global variables have been placed in the unit UFloaters, which is included by both the main program file, FloaterDemoPascal.p, and the unit UFloatRoutines.

The interface section

The constant declaration block

Lines 21-31 establish constants relating to menu IDs and menu item numbers. Lines 33-43 establish constants relating to menu bar, window, alert, dialog, picture, and sound resources. Lines 45-46 establish constants which will be used to identify a particular window as being of the document type or the floating type. Line 48 defines kMaxLong as the maximum possible long value.

The type declaration block

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. Lines 54-58 define ActivateProcPtr as a pointer to a function that takes a WindowPtr and a Boolean as parameters and returns nothing.

Lines 60-67 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.

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.

The implementation section

The procedure DoOpenFloatingWindows

DoOpenFloatingWindows opens the two floating windows.

Lines 158-159 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 161 gets the address of the application-defined routine for activating/deactivating the Tools floating window. Line 162 opens the Tools floating window by calling the special application-defined function for opening windows in a floating windows environment. At Line 170, the appropriate 'PICT' resource for the Tools window is loaded. The call to SetWindowPic at Line 171 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 173-183 repeat this process for the Colours window. Line 185 re-highlights the Tools window. Both windows are now highlighted and active.

The procedure 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 206), the program's event handler is called (Line 207). If the event belongs to a dialog window (Line 208), DialogSelect is called at Line 209 to process the event. When DialogSelect returns, Lines 211-212 dispose of the dialog.

The procedure 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.

The procedure DoMouseDown

DoMouseDown further processes mouse-down events.

If the mouse-down was in the content region of a window (Line 280), the following occurs. If the front window is not a dialog window (Line 282), an application-defined function is called at Line 283 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 284), the system alert sound is played (Line 285); 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 286-287).

If the mouse-down was in the title bar of a window (Line 290), 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 292-293); otherwise, the special application-defined function which replaces DragWindow in a floating window environment is called (Line 295).

If the mouse-down was in the close box (Line 298), and if TrackGoAway returns true (Line 299), the following occurs. A value representing the window type (floating or document) is retrieved from the window's document record (Lines 301-302). If the window is a floating window (Line 303), the special application-defined function which replaces HideWindow in a floating window environment is called (Line 305) and the menus are adjusted to remove the checkmark from the relevant Floaters menu item (Line 306). If the window is a document window, the application-defined function which closes document windows is called (Lines 308-309).

The handling of mouse-downs in the grow box and the zoom box (Lines 313-333) is as for handling in a non-floating windows environment. Note that the third parameter in the call to ZoomWindow at Line 331 must be false so that the window is not brought to the front.

The procedure DoUpdate

DoUpdate further processes update events. Recall that, because of the SetWindowPic calls at Lines 171 and 183, the floating windows will not receive update events. Accordingly, the only windows redrawn by this function are document windows (Line 352).

The procedure DoOSEvent

DoOSEvent further processes Operating System events.

In the case of a resume event, the special application-defined procedure which handles resume events in a floating windows environment is called (Line 369). In the case of a suspend event, the special application-defined procedure which handles suspend events in a floating windows environment is called (Line 374). As usual, the global variable gInBackground is set appropriately although, in this particular demonstration program, it actually has no part to play.

The procedure 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.

The procedure 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 449-450). When the alert box is dismissed, the special application-defined function which performs floating window and document window activation is called (Line 451).

The procedure DoFileMenu

DoFileMenu further processes File menu choices.

The procedure DoFloatersMenu

DoFloatersMenu further processes Floater menu choices.

Lines 509-512 assign the pointer to the floating window associated with the chosen item to a variable. Line 514 determines whether that floating window is currently visible. If it is not visible, the special application-defined procedure which replaces ShowWindow in a floating windows environment is called (Lines 516-517); otherwise, the special application-defined procedure which replaces HideWindow in a floating windows environment is called (Line 519).

Line 521 adjusts the menus to include/remove the checkmark in/from the relevant Floaters menu item, as appropriate.

The procedure 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 534-535); otherwise, Line 537 gets a pointer to the application-defined function for activating/deactivating document windows, the special application-defined procedures which open windows and replace ShowWindow in a floating windows environment are called (Lines 538-540) and the global variable which keeps track of the number of open windows is incremented (Line 541).

The procedure 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 554 attempts to get a pointer to the front document window. If nil is returned by FindFrontNonFloatingWindow, no document windows are open. If a non-nil value is returned, the special application-defined procedure which closes windows in a floating windows environment is called (Line 558). Line 559 decrements the global variable which keeps track of the number of open windows.

The procedure 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.

The procedure 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.)

The procedure DoOpenFindDialog

DoOpenFindDialog responds to the choice of the Find… item in the File menu. Before opening the dialog, a call is made to the special application-defined procedure which handles window deactivation in a floating windows environment (Line 616). The menus are then adjusted as appropriate in the presence of an open modal dialog.

The procedure DoDisposeFindDialog

DoDisposeFindDialog is called when the user dismisses the Find dialog. After closing the dialog (Lines 631-632), a call is made to the special application-defined procedure which handles window activation in a floating windows environment (Line 633). The menus are then re-adjusted (Line 720).

The procedure 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.

The procedure ToolsActivateHandler

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

Line 684 sets Tools floating window's graphics port as the current graphics port.

Lines 686-689 determine which 'PICT' resource will be loaded by the GetPicture call at Line 794. 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 694), a copy is made of the rectangle in the picture record's picFrame field (Line 695) and that rectangle is offset so that the left and top fields are both 0. Lines 697-698 erase the port rectangle if Color QuickDraw is not present. Line 699 draws the picture and Line 700 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.

The procedure 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.

The procedure InvalidateScrollBarArea

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

FloaterDemoPascal.p

The main program block

The main function firstly initialises the system software managers (Line 808). At Lines 812-821, 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 825-835 set up the menus and Lines 839-842 opens the two floating windows and sets a checkmark in their respective Floaters menu items. The main event loop is entered at Line 846.

The unit UFloatRoutines.p

UFloatRoutines.p 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.

The function 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 940 assigns memory for the window's document record. Lines 942-946 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 948), the handle to the document record is assigned to the window record's refCon field (Line 950) and the pointer to the window's activation function is assigned to the activateHandler field of the window's document record (Lines 951-952). If the window is a floating window (Line 954), the windowType field of the window's document record is assigned a value representing that window kind and the window is highlighted (Lines 956-957). If the window is not of the floating kind (Line 959), 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 963 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 965-966), otherwise the window is brought to the front of all windows (Line 966).

If the window was not successfully opened, the document record is disposed of (Line 973). Line 975 returns the result of the call to GetNewWindow/GetNewCWindow.

The procedure 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 984), a call is made to the procedure which replaces HideWindow so that the next document window in the list (if any) is activated (Line 985). Line 986 then removes the window from the screen and the window list, Line 987 disposes of the window's document record, and Line 988 disposes of the window's window record.

The procedure 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 1002), Line 1004 records that fact and Line 1005 gets the pointer to the current front window, which will be a floating window. If the window clicked was a document window (Line 1007), Line 1008 records that fact, Line 1009 gets a pointer to the first document window in the list, and Line 1010 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 1013), and if the window clicked is a floating window (Line 1015), that window is brought to the very front of the list (Line 1016). If the window clicked is a document window (Line 1017), and if there are no floating windows (Line 1018), 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 1020), an application-defined procedure 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.

The procedure 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 1041-1042).

Line 1044 gets a pointer to the frontmost window. If this window is not a floating window, a variable is set to record that fact (Lines 1045-1046).

Line 1048 gets a pointer to the first document window.

Line 1050 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 1052), and if the next window in the list (if any) is a floating window (Line 1056) the hidden window is moved behind that window (Lines 1058-1060). (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 1063), if it is the front document window (Line 1064), and if there is another document window behind it (Lines 1066-1068), the following occurs. Lines 1070-1071 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 1073-1074), Line 1075 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 (Line 1077). The new front document window is then activated (Line 1079).

The procedure 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 1101-1102).

Line 1104 gets the type of the specified window. If the (currently hidden) window is a document window (Line 1106), and if that window is the front document window (Lines 1108-1109), the following occurs. If there is a document window behind the specified window (Line 1111), that window is deactivated (Line 1112). 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 1135-1140) cause the window's activation function to be called.

If the specified (currently hidden) window is a floating window, Line 1119 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 1135-1140) cause the window's activation function to be called.

Regardless of what has gone on to this point, Line 1133 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 procedure for the specified window is retrieved from the window's window record and that procedure is called to complete the activation process, the true parameter advising that procedure that activation-related actions, as opposed to deactivation-related actions, are to be performed (Lines 1135-1140).

The procedure 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 1167) tests whether the mouse button has remained down since the last mouseDown event. If it has, the following occurs.

Lines 1169-1173 adjust the top of the dragging rectangle so that it is below the menu bar. Lines 1175-1177 save the current graphics port, and set the window manager port as the current graphics port. Line 1179 sets the clipping region to the region below the menu bar.

Lines 1181-1183 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 1189 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 1191 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 1194-1195 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 1197, 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 1200 sets the port saved at Line 1175 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 1202), the value in both words is retrieved at Lines 1204-1205. If the mouse was not outside the slop rectangle (Line 1207), the new horizontal and vertical global coordinates are calculated (Lines 1209-1213) and passed as a parameter to the MoveWindow call at Line 1215, 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 1220), the call to NewSelectWindow at Line 1221 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).

The procedure HandleSuspendEvent

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

Line 1237 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 1239), the function simply returns.

The first time through the while loop entered at Line 1242, 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 1245-1246). If the window is visible, it is hidden without affecting the front-to-back ordering of the open windows (Lines 1247-1248). Line 1249 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 1252-1257).

> Note that the call to DrawGrowIcon at Line 1255 should be removed if your do not require your document windows to be resizeable.

The procedure 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 1270 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 1272), the function simply returns.

The first time through the while loop entered at Line 1275, 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 1278). If the window was visible, it is shown without affecting the front-to-back ordering of the open windows and then activated (Lines 1279-1283). Line 1284 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 1287-1292).

> Note that the call to DrawGrowIcon at Line 1290 should be removed if your do not require your document windows to be resizeable.

The procedure 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 1306 gets a pointer to the frontmost window. Line 1307 attempts to get a pointer to the first document window. If at least one document window exists (Line 1308), Line 1309 attempts to get a pointer to the second document. The variable secondDocumentWindowPtr now contains either a pointer or nil. The while loop entered at Line 1312 walks the window list up to and including the first document window, deactivating all visible windows.

The procedure 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 1337-1341 determine whether this program is in the background. If it is, any visible floating windows are hidden (Lines 1343-1344).

If the program is not in the background (Line 1345), Lines 1346-1357 activate all floating windows and the first document window, using the same list-walking methodology as DeactivateFloatersAndFirstDocWin.

The procedures 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 1389), retrieves the pointer to the window's activation routine from the window's document record (Lines 1390-1391), and calls that routine (Line 1393).

FindFrontNonFloatingWindow, etc

FindFrontNonFloatingWindow is the first of those functions and procedures which support the main UFloatRoutines.p routines 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.

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.

WDEFPascal.p

WDEFPascal.p 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.

The constant declaration block

Lines 1635-1637 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.

The type declaration block

Rather than use global variables (as in the control definition function CDEF1 at Chapter 19), the approach used here is to store "global" data in the fields of a Globals record (Lines 1662-1782), the handle to which will be assigned to the dataHandle field of the window record.

gColorQuickDrawPresent 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. gCloseEnabledHdl to gCheckPatternHdl will be assigned handles to the three colour icons. gWhite to gBlack will be 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.

The function 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,

Although no global variables are used in this program, Lines 1712 and 1748 remain necessary. In this program, the Line 1738 call is required so as to prevent Gestalt returning -5551 (undefined selector).

Line 1714 saves the current pen location, size, mode and pattern. Line 1715 saves the current graphics port. If Color QuickDraw is present, the WDEF-defined procedure SyncPorts is called (Lines 1717-1719). (As will be seen, gColourQuickDrawPresent is set in the WDEF's response to the wNew message.)

Line 1721 initialises the variable which contains the value to be returned by the WDEF.

Lines 1723-1741 switch according to the value received in the message parameter. Note that, at Lines 1729-1732, action is taken in response to the wDraw message only if the specified window is currently visible.

Lines 1744-1745 restore the saved pen state and graphics port. Line 1774 restores the A4 register's value saved at Line 1738.

Line 1746 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.

The procedure DoInitMessage

DoInitMessage is called when the wNew message is received.

Firstly, a handle to a Globals record is obtained, and, if this was successful, various fields are initialised (Lines 1764-1803).

At Lines 1805-1814, the gColorQuickDrawPresent field is set to true if Color QuickDraw is present and, if Color QuickDraw is present, the gColourDisplay field is set to true if the pixel depth of the main device is greater than 1.

Lines 1816-1817 initialise the global variable gBlackPattern.

Lines 1819-1824 load the three colour icons.

Line 1826 initialises the gToggle field, which will be used in toggling the close box between the normal and highlighted states.

Line 1828 assigns the handle to the Globals record to the window record's dataHandle field.

The procedure DoDrawMessage

DoDrawMessage is called when the wDraw message is received.

Lines 1843-1844 clears the high word of the param parameter because, in the case of the wDraw message, the high word may contain undefined data.

Lines 1816-1862 branch 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 1848-1853) or the close box to be toggled to its opposite state (Lines 1855-1857). 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.

The function 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 1876-1877 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 1881-1882) and the function exits. If not, Line 1884 checks whether the mouse-down was in the structure region. If it was, and if the window has a close box (Line 1886), the close box rectangle is retrieved (Line 1888). If the mouse-down was in the close box, wInGoAway is returned and the function

exits (Lines 1891-1892), otherwise wInDrag is returned and the function exits (Lines 1896-1897).

If none of these checks prove positive, wNoHit is returned (Line 1900).

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.

The procedure 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 1915 calls a WDEF-defined procedure to get a rectangle equivalent to the window's port rectangle converted to global coordinates. Line 1916 sets the window's content region to equate to this rectangle.

Line 1918 calls a WDEF-defined procedure 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 1919 sets the window's structure region to equate to this rectangle. The rectangle is then offset 1 pixel down and to the right (Line 1920) and the top and left values are further increased (by one pixel) at Lines 1921-1922. Line 1923 then turns this rectangle into a region and Line 1924 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.

The procedure 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 1944-1945 save the current foreground and background colours. Lines 1946-1948 set the pen size, pattern, and transfer mode.

Lines 1952-2017 execute only if the window is currently active (Line 1950). These lines draw a black frame one pixel outside the content rectangle (Lines 1952-1956) and then base all drawing on the structure rectangle (Line 1958). Lines 1960-1963 paint the title bar to the right of the close box rectangle. Lines 1965-1974 use the checkered colour icon to draw the checkered pattern in the title bar. Lines 1976-1977 use the colour icon for the close box in the normal state to draw the close box. Lines 1979-2017 then draw the remainder of the window frame.

Lines 2019-2036 execute only if the window is currently inactive (Line 2018), and draw the window frame in the inactive state.

Lines 2038-2039 restore the foreground and background colours saved at Lines 1944-1945.

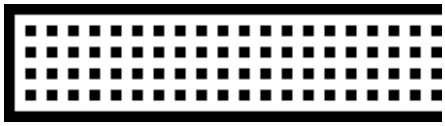
The procedure DrawWindowMono

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.

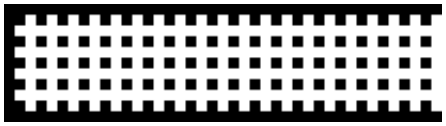
Lines 2057-2059 set the pen size, pattern, and transfer mode.

Lines 2063-2112 execute only if the window is currently active. These lines draw a black frame one pixel outside the content rectangle (Lines 2066-2069) and then base all drawing on the structure rectangle (Line 2071). Lines 2073-2076 frame the structure rectangle and add the drop shadows. Lines 2078-2080 erase the title bar area. Lines 2082-2106 set the pen pattern to a checkered pattern before drawing the title bar. Line 2108 restores the pen pattern to a black pattern and Lines 2110-2111 draw the close box.

Some further explanation of Lines 2085-2101 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 2090-2093 are central in this respect.

If the window is currently inactive (Line 2113), the title bar is simply erased (Lines 2114-2116).

The procedure ToggleGoAway

ToggleGoAway is called when the wDraw message is received with wInGoAway in the param field. It first reverses the value in the Boolean gToggle (Line 2131) and then calls the appropriate WDEF-defined procedure to draw the close box in the normal state or the highlighted state, depending on the value in the gToggle variable (Lines 2133-2136).

The procedure DrawGoAwayBox

DrawGoAwayBox draws the close box in its normal state, either in colour (Lines 2153-2154) or black-and-white if it is not (Lines 2155-2159).

The procedure DrawGoAwayBoxPressed

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

The procedure 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.

The procedure GetContentRect

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

The procedure 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..

The procedure 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 2266 gets a pointer to the Window Manager port. Line 2267 gets a pointer to the Window Manager color port and Line 2268 sets that port as the current port.

Line 2270 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 2271 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 2273-2274 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 in WDEFPascal.p, follow the same general procedure as is described for LDEFs in the Demonstration Program Comments at Chapter 18 — Lists and Custom List Definition Functions.