4B

MORE ON WINDOWS — MAC OS 8.5 WINDOW MANAGER

Includes Demonstration Program Windows2

Introduction

The Mac OS 8.5 Window Manager introduced the following:

- Support for:
    - Floating windows.

> **Note**
>
> Although system support for floating windows was introduced with the Mac OS 8.5 Window Manager, Apple subsequently advised that there were significant bugs in the window activation area and that the Application Programming Interfaces (APIs) relating to floating windows should not be used until further notice.
>
> These bugs were eliminated in Mac OS 8.6. The upshot of all this is that, if your application uses floating windows, and if it is required to run under Mac OS 8.5 or earlier, your application will itself have to include the code necessary to support floating windows. (See Chapter 21 — Floating Windows — Mac OS 8.5 and Earlier.)

    - Window proxy icons.
    - Window path pop-up menus.
    - Transitional window animations and sounds.
- New functions for:
    - Creating and storing windows.
    - Accessing window information.
    - Zooming, moving, re-sizing, and positioning windows.
    - Associating data with a window.

- Adding and removing rectangles and regions to and from a window's update region.

- Setting the colour or pattern of a window's content region.

Floating Windows

Floating windows are windows that 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.

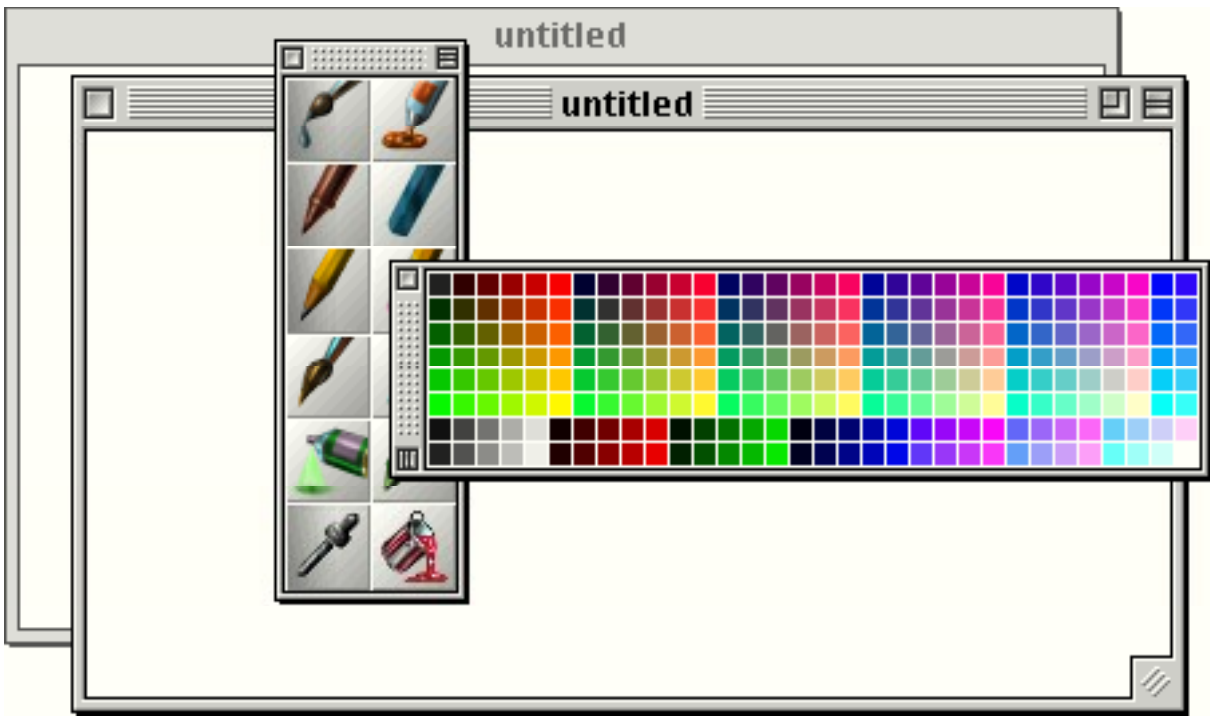


FIG 1 - FLOATING WINDOWS - EXAMPLES

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, the correct front-to-back ordering is as follows:

- Help balloons.

- Menus.

- System windows.[1]

- Modal and movable modal dialog and alert boxes.

- Floating windows.

- Document windows and modeless dialog boxes.

In terms of front-to-back ordering, floating windows, unlike document windows, are all basically equal. Unless they actually overlap each other, there is no visual cue of any front-to-back ordering as there is with normal windows (see Fig 1). Because of this equality, floating windows almost always appear in the active state. The exception is

[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.

when a modal or movable modal dialog or alert box is presented to the user. When this occurs, the appearance of all floating windows changes to reflect the inactive state.

Window Activation

Window activation was the most significant aspect of implementing system support for floating windows. The pre-Mac OS 8.5 Window Manager is based on the principle that, at any one time, there can be one, and only one, active window. This "one active window" rule, however, cannot apply in the case of an application which uses floating windows. (See Fig 1, in which the two floating windows and the frontmost document window are active at the same time.) Accordingly, applications which require floating windows, and which are required to run under Mac OS 8.5 or earlier, must include code which, in effect, subverts the normal window activation activities of the Window Manager. (See Chapter 21 — Floating Windows — Mac OS 8.5 and Earlier.)

In Mac OS 8.5 and later, support for floating windows is built into the Window Manager, meaning that, amongst other things, the system now supports the activation of more than one window at a time.

Floating Window Types

The sixteen available window types for floating windows are shown at Figs 4 and 5 at Chapter 4 — Windows.

Opening, Closing, Showing, and Hiding Floating Windows

Floating windows may be created using the Mac OS 8.5 function CreateNewWindow (see below) with the constant kFloatingWindowClass passed in the windowClass parameter.

Floating windows should be created at application launch and should remain open until the application is closed. However, your application should provide the user with a means to hide or show each individual floating window as and when required. Ordinarily, it should do this by providing items in an appropriate menu which allow the user to toggle each floating window between the hidden and showing states.

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.

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 application's floating windows should be shown again only when the application receives a subsequent resume event.

Mac OS 8.5 Functions Relating to Floating Windows

The following Mac OS 8.5 functions are relevant to floating windows:

Function	Description
InitFloatingWindows	Initialises the Window Manager and enables automatic front-to-back display ordering of all your application's windows. If your application uses floating windows, you must call InitFloatingWindows in lieu of InitWindows. When InitFloatingWindows has been called, each of your application's windows is sorted into one of three window display layers: modal, floating, and document. For windows created with the Mac OS 8.5 function CreateNewWindow (see below), sorting is based on window class (see below). For windows created using the pre-Mac OS 8.5

	functions, the sort order is based on window definition ID.
HideFloatingWindows	Hides an application's floating windows.
ShowFloatingWindows	Shows an application's floating windows.
AreFloatingWindowsVisible	Indicates whether an application's floating windows are visible.

Window Proxy Icons

Window proxy icons are small icons displayed in the title bar of document windows. Ordinarily, a specific document file is associated with a specific window, and the proxy icon serves as a proxy for the document file's icon in the Finder.

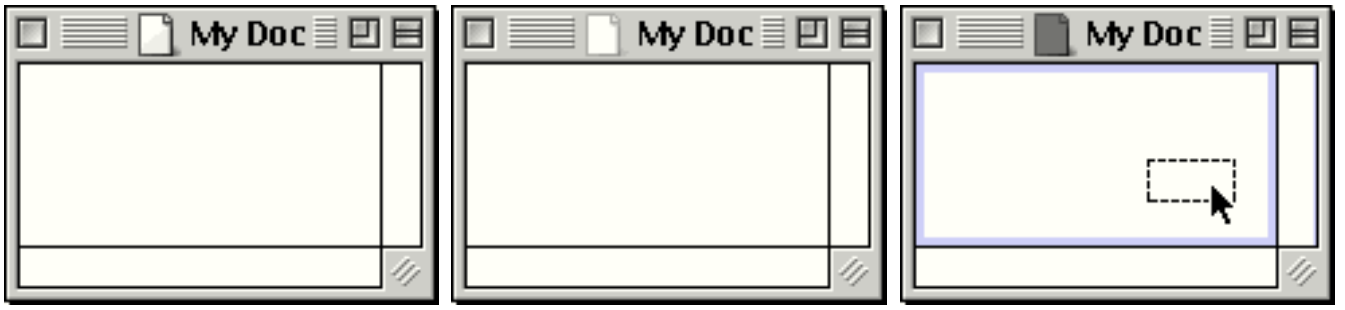
Proxy icons:

- May be dragged, in the same way that the document's icon in the Finder may be dragged, so as to move or copy the document file.

- Provide visual feedback to the user on the current state of the document. For example, when the document has unsaved changes, your application should cause the proxy icon to be displayed in the disabled state, thus preventing the user from dragging it. (Unsaved documents should not be capable of being moved or copied.)

- Provide visual feedback to the user indicating that the document window is a valid drag-and-drop target. In this case, your application should cause the proxy icon to appear in the highlighted state.

Fig 2 shows a typical window proxy icon for a document in the enabled, disabled, and highlighted states.



PROXY ICON IN ENABLED STATE (DOCUMENT HAS NO UNSAVED CHANGES) **PROXY ICON IN DISABLED STATE (DOCUMENT HAS UNSAVED CHANGES)** **PROXY ICON IN HIGHLIGHTED (WINDOW IS VALID DRAG-&-DRO**

FIG 2 - WINDOW PROXY ICONS

At Fig 2, note that, in the drag and drop operation depicted at the right, the window's content area is highlighted along with the proxy icon. Applications typically call the Drag Manager function ShowDragHilite to indicate, with this highlighting, that a window is a valid drag-and-drop target. Under Mac OS 8.5 and later, ShowDragHilite and HideDragHilite highlight and unhighlight the proxy icon as well as the content area.

Changing the State of a Proxy Icon

Applications typically keep track of the modification state of a document so as to, for example, inform users that they has made changes to the document which they might wish to save before closing the document's window. When a document has unsaved changes, your application should call SetWindowModified with true passed in the modified parameter to cause the proxy icon to appear in the disabled state. When the changes have been saved, your application should call SetWindowModified with false passed in the modified parameter to cause the proxy icon to appear in the enabled state.

Handling Mouse-Down Events in a Window Proxy Icon

When a mouse-down event occurs in your application's window, and when FindWindow returns the inProxyIcon result code, your application should simply call TrackWindowProxyDrag. TrackWindowProxyDrag handles all aspects of the drag process while the user drags the proxy icon.

File Synchronisation Function

It is always possible that, while a document file is open, the user may drag its Finder icon to another folder (including the Trash) or change the name of the file via the Finder icon. The application itself has no way of knowing that this has happened and will assume, unless it is informed otherwise, that the document's file is still at its original location with its original name. For this reason applications often include a frequently-called **file synchronisation function** which synchronises the application with the actual current location (and name) of its currently open document files.

A document's proxy icon is much more prominent to the user than the document's Finder icon. Thus, when proxy icons are used, there is an even a greater possibility that the user will move the file represented by the proxy icon to a different folder while the document is open. The provision of a file synchronisation function is therefore imperative when proxy icons are implemented.

File synchronisation functions should be called after every call to WaitNextEvent and, for each of the application's document windows, should update the application's internal data structures to match that of the document file as it exists on disk. The function should also ensure that, where necessary, the name of the document window is changed to match the current name of the document file on disk and close the document window if the document file has been moved to the Trash folder.

Mac OS 8.5 Functions Relating to Window Proxy Icons

The following Mac OS 8.5 functions are relevant to window proxy icons:

Function	Description
SetWindowProxyCreatorAndType	Sets the proxy icon for a window that lacks an associated file. A new, untitled window needs a proxy icon to maintain visual consistency with other windows. Call this function when you want to establish a proxy icon for the window but the window's data has not yet been saved to a file.
SetWindowProxyFSSpec	Associates a file with a window using a file system specification (FSSpec) structure, thus establishing a proxy icon for the window.
GetWindowProxyFSSpec	Obtains a file system specification (FSSpec) structure for the file that is associated with a window.
SetWindowProxyAlias	Associates a file with a window using a handle to an AliasRecord structure, thus establishing a proxy icon for the window.
GetWindowProxyAlias	Obtains alias data for the file associated with the window.
SetWindowProxyIcon	Overrides the default proxy icon for a window.
GetWindowProxyIcon	Obtains a window's proxy icon.
RemoveWindowProxy	Dissociates a file from a window.
TrackWindowProxyDrag	Handles all aspects of the drag process when the user drags a proxy icon.

Note that SetPort should be called to set the relevant window's colour graphics port as the current port before calling SetWindowProxyCreatorAndType, SetWindowProxyFSSpec, SetWindowProxyAlias, and SetWindowProxyIcon.

Window Path Pop-Up Menus

If your application supports window path pop-up menus, when the user presses the Command key and clicks a window's title, your window displays a pop-up menu containing a standard file system path. The pop-up menu allows the user to open windows for folders along the file system path. An example of a window path pop-up menu is shown at Fig 3.

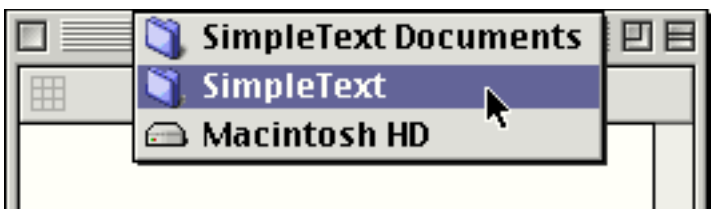


FIG 3 - WINDOW PATH POP-UP MENU

Displaying and Handling a Window Path Pop-Up Menu

The window title includes both the proxy icon region and part of the drag region. Your application must be prepared to respond to a Command-click in either region by displaying a window path pop-up menu.

When FindWindow returns the inProxyIcon part code, and TrackWindowProxyDrag returns errUserWantsToDragWindow, your application should proceed on the assumption that the inDrag part code was returned by FindWindow.

When FindWindow returns the inDrag part code, your application should call IsWindowPathSelectClick to determine whether the mouse-down event should activate the window path pop-up menu. If IsWindowPathSelectClick returns true, WindowPathSelect should be called to display the menu.

If the user chooses a menu item for a folder, your application must ensure that the associated window is visible by calling an application-defined function which makes the Finder the frontmost process.

Transitional Window Animation and Sounds

Prior to Mac OS 8.5, the Window Manager supported the playing of a sound to accompany the transitional animation that occurs when a user clicks a window's collapse box. Mac OS 8.5 added support for animation and sounds to accompany the hiding and showing of windows.

The Mac OS 8.5 Window Manager function TransitionWindow may be used in lieu of the older functions HideWindow and ShowWindow to hide and show windows. TransitionWindow causes a transitional animation to be displayed, a transitional sound to be played, and the necessary update and activate events to be generated.

Creating and Storing Windows

Mac OS 8.5 provides the following functions for creating and storing windows:

Function	Description
CreateNewWindow	Creates a window from parameter data.
CreateWindowFromResource	Creates a window from 'wind' resource data.
CreateWindowFromCollection	Creates a window from collection data.
StoreWindowIntoCollection	Stores data describing a window into a collection.

Use of the last three of these functions requires a basic understanding of **collections**, **flattened collections** and **'wind' resources**.

Collections, Flattened Collections, and 'wind' Resources

Collections

A **collection object** (or, simply, a collection) is an abstract data type, defined by the Collection Manager, that allows you to store multiple pieces of related information.

A collection is like an array in that it contains a number of individually accessible items. However, unlike an array, a collection allows for a variable number of data items and variable-size items. A collection is also similar to a database, in that you can store information and retrieve it using a variety of search mechanisms.

The internal structure of a collection is private. This means that you must store information into a collection and retrieve information from it using Collection Manager functions.

Using the function StoreWindowIntoCollection, your application can store a window into a collection. This applies to any window, not just those created using Mac OS 8.5 Window Manager functions. You can also store data associated with the window (for example, text) into the same collection. This provides a quick and easy way for your application to save a simple document.

Using the Mac OS 8.5 function CreateWindowFromCollection, you can create a window from collection data. Note that CreateWindowFromCollection creates the window invisibly. After creating the window, you must call the function TransitionWindow to display the window.

Flattened Collections

Using the Collection Manager, you application can create a flattened collection from a collection. A flattened collection is a stream of address-independent data.

The 'wind' Resource

The 'wind' resource consists of an extensible flattened collection. Using the Resource Manager, your application can store a flattened collection, consisting of a window and its data, into a 'wind' resource.

Using the Mac OS 8.5 function CreateWindowFromResource, you can create a window from a 'wind' resource. Note that CreateWindowFromResource creates the window invisibly. After creating the window, you must call the function TransitionWindow to display the window.

The CreateNewWindow Function

The Mac OS 8.5 function CreateNewWindow creates a window based on the class and attributes you specify in the windowClass and attributes parameters. The following constants may be passed in these parameters.

Window Class Constants

Constant	Value	Description
kAlertWindowClass	1L	Alert box window.
kMovableAlertWindowClass	2L	Movable alert box window.
kModalWindowClass	3L	Modal dialog box window.
kMovableModalWindowClass	4L	Movable modal dialog box window.
kFloatingWindowClass	5L	Floating window.

		If your application assigns this constant to a window and calls the function InitFloatingWindows, the Window Manager ensures that the window has the proper floating behaviour. Supported with Mac OS 8.6 and later.
kDocumentWindowClass	6L	Document window or modeless dialog box window. The Window Manager assigns this class to pre–Mac OS 8.5 Window Manager windows.

Window Attribute Constants

Constant	*Bit*	*Description*
kWindowNoAttributes	0L	No attributes.
kWindowCloseBoxAttribute	1L << 0	Has close box.
kWindowHorizontalZoomAttribute	1L << 1	Has horizontal zoom box.
kWindowVerticalZoomAttribute	1L << 2	Has vertical zoom box.
kWindowFullZoomAttribute	kWindowVerticalZoomAttribute | kWindowHorizontalZoomAttribute	Has full zoom box.
kWindowCollapseBoxAttribute	1L << 3	Has a collapse box.
kWindowResizableAttribute	1L << 4	Has size box.
kWindowSideTitlebarAttribute	1L << 5	Has side title bar. This attribute may be applied only to floating windows.
kWindowNoUpdatesAttribute	1L << 16	Does not receive update events.
kWindowNoActivatesAttribute	1L << 17	Does not receive activate events.
kWindowStandardDocumentAttributes	kWindowCloseBoxAttribute | kWindowFullZoomAttribute | kWindowCollapseBoxAttribute | kWindowResizableAttribute	Has standard document window attributes, that is, close box, full zoom box, collapse box and size box.
kWindowStandardFloatingAttributes	kWindowCloseBoxAttribute | kWindowCollapseBoxAttribute	Has standard floating window attributes, that is, close box and collapse box.

Note that CreateNewWindow creates the window invisibly. After creating the window, you must call the function TransitionWindow to display the window.

Accessing Window Information

Mac OS 8.5 includes the following functions for accessing window information:

Function	*Description*
GetWindowClass	Obtains the class of a window.
GetWindowAttributes	Obtains the attributes of a window.
IsValidWindowPtr	Reports whether a pointer is a valid window pointer.
FrontNonFloatingWindow	Returns a pointer to the application's frontmost window that is not a floating window.

Zooming, Moving, Resizing, and Positioning Windows

Zooming Windows

The Mac OS 8.5 function ZoomWindowIdeal may be used instead of ZoomWindow to zoom a window, the advantage being that ZoomWindowIdeal zooms the window in accordance with the following human interface guidelines relating to a window's standard state:

- A window should move as little as possible when zooming between the user state and standard state, to avoid distracting the user.
- A window in its standard state should be positioned so that it is entirely on one screen.
- If a window straddles more than one screen in the user state, when it is zoomed to the standard state it should be zoomed to the screen that contains the largest portion of the window’s content region.
- If the ideal size for the standard state is larger than the destination screen, the dimensions of the standard state should be that of the destination screen, minus a few pixels of boundary. If the destination screen is the main screen, space should also be left for the menu bar.
- When a window is zoomed from the user state to the standard state, the top left corner of the window should remain anchored in place; however, if the standard state of the window cannot fit on the screen with the top left corner anchored, the window should be “nudged” so that the parts of the window in the standard state that would fall offscreen are, instead, just onscreen.

The ZoomWindowIdeal function calculates the window's ideal standard state, and updates the window's user state independently of the WStateData structure. (Previously, the window definition function was responsible for updating the user state.)

When ZoomWindowIdeal is used, the Mac OS 8 function IsWindowInStandardState must be used to determine the appropriate part code (inZoomIn or InZoomOut) to pass in ZoomWindowIdeal's partCode parameter.

The following two additional Mac OS 8.5 functions relating to window zooming allow your application to access the window's user-state in a Carbon-compliant manner.

Function	Description
SetWindowIdealUserState	Sets the size and position of a window in its user state. The size and position of the window are specified in global coordinates in the userState parameter.
GetWindowIdealUserState	Obtains the size and position of a window in its user state. On return, the userState parameter contains the size and position in global coordinates.

Ordinarily, your application does not need to use these two functions. They are supplied for the sake of completeness.

Moving Windows

When your application wishes to move a window for a reason other than a user-instigated drag, it should use the Mac OS 8.5 function MoveWindowStructure or the earlier function MoveWindow.

MoveWindow repositions a window's content region, whereas MoveWindowStructure repositions a window's structure region. The introduction of the MoveWindowStructure function arises from

the fact that, under the Appearance Manager, the size and shape of a window's frame may vary from appearance to appearance (see Chapter 6 — The Appearance Manager). This means that the total dimensions of the window (that is, the window's structure region) may also vary, causing the window's spatial relationship to the rest of the screen to change.

The Mac OS 8.5 function SetWindowBounds provides a means to set the size of a window in addition to simply repositioning it. The size and position of the window are specified in a rectangle passed in the globalBounds parameter. In addition, you may specify whether this rectangle represents the bounds of the content region or the bounds of the structure region by passing either kWindowContentRgn or kWindowStructureRgn in the regionCode parameter. The sister Mac OS 8.5 function GetWindowBounds obtains the size and position of the bounding rectangle of the specified window region.

Resizing Windows

The Mac OS 8.6 function ResizeWindow moves a grow image of the window's edges around the screen, following the user's cursor movements, and handles all user interaction until the mouse button is released. Unlike the function GrowWindow, there is no need to follow this call with a call to SizeWindow. Once resizing is complete, ResizeWindow draws the window in its new size.

> **Note**
>
> ResizeWindow is supported only under Mac OS 8.6 and later.

ResizeWindow informs your application of the new window bounds, so that, if necessary, your application can respond to any changes in the window's position. (This latter possibility arises from the fact that some appearances may allow the window to be resized from any corner, not just the bottom right, as a result of which the window may move on the screen and not simply change size.)

Positioning Windows

Generally speaking, a new window should be placed on the desktop where the user expects it to appear. For new document windows, this usually means just below and to the right of the last document window in which the user was working, although this is not necessarily the case on computers with multiple monitors.

The Mac OS 8.5 function RepositionWindow allows you to position a window relative to another window or a display screen. The required window positioning method may be specified by passing one of the following constants in the method parameter.

Window Positioning Constants

Constant	Value	Description
kWindowCenterOnMainScreen	0x00000001	Centre on the screen that contains the menu bar.
kWindowCenterOnParentWindow	0x00000002	Centre on the parent window. If the window to be centred is wider than the parent window, its left edge is aligned with the parent window's left edge.
kWindowCenterOnParentWindowScreen	0x00000003	Centre on the screen containing the parent window.
kWindowCascadeOnMainScreen	0x00000004	Place the window just below the menu bar at the left edge of the main screen. Place subsequent windows relative to the first window such that

		the frame of the preceding window remains visible behind the current window.
kWindowCascadeOnParentWindow	0x00000005	Place the window a distance below and to the right of the upper-left corner of the parent window such that the frame of the parent window remains visible behind the current window.
kWIndowCascadeOnParentWindowScreen	0x00000006	Place the window just below the menu bar at the left edge of the screen containing the parent window. Place subsequent windows on the screen relative to the first window such that the frame of the preceding window remains visible behind the current window.
kWindowAlertPositionOnMainScreen	0x00000007	Centre the window horizontally, and position it vertically on the screen that contains the menu bar such that about one-fifth of the screen is above it.
kWindowAlertPositionOnParentWindow	0x00000008	Centre the window horizontally, and position it vertically such that about one-fifth of the parent window is above it.
kWindowAlertPositionOnParentWindowScreen	0x00000009	Centre the window horizontally, and position it vertically such that about one-fifth of the screen containing the parent window is above it.

These constants should not be confused with the pre-Mac OS 8.5 positioning specification constants (see Chapter 4 — Windows), and should not be used where those older constants are required (for example, in 'WIND', 'DLOG', and 'ALRT' resources, and in the StandardAlert function).

Associating Data With Windows

The pre-Mac OS 8.5 function SetWRefCon allows your application to associate a pointer to data with a pointer to a window. An alternative method of associating data with windows is to use the standard mechanism provided by the Mac OS 8.5 Window Manager. (Both methods, incidentally, are Carbon-compliant.)

The Mac OS 8.5 Window Manager provides the following functions relating to associating data with windows:

Function	Description
SetWindowProperty	Associates an arbitrary piece of data with a window.
GetWindowProperty	Obtains a piece of data associated with a window.
GetWindowPropertySize	Obtains the size of a piece of data associated with a window.
RemoveWindowProperty	Removes a piece of data associated with a window.

Adding To and Removing From the Update Region

The Mac OS 8.5 Window Manager provides enhanced functions for manipulating the update region. Unlike their pre-Mac OS 8.5 counterparts, the new functions allow the window on which they operate to be explicitly specified, meaning that they do not require the graphics port to be set prior to their use.

The following are the Mac OS 8.5 functions for manipulating the update region:

Function	Description
InvalWindowRect	Adds a rectangle to the window's update region.
InvalWindowRgn	Adds a region to the window's update region.
ValidWindowRect	Removes a rectangle from the window's update region.
ValidWindowRgn	Removes a region from the window's update region.

Setting Content Region Colour and Pattern

The Mac OS 8.5 Window Manager provides the following functions for setting the colour or pattern of a window's content region:

Function	Description
SetWindowContentColor	Sets the colour to which a window's content region is redrawn on receipt of an update event.
GetWindowContentColor	Obtains the colour to which a window's content region is redrawn.
SetWindowContentPattern	Sets the pattern to which a window's content region is redrawn on receipt of an update event.
GetWindowContentPattern	Obtains the pattern to which a window's content region is redrawn.

These functions do not affect the colour graphics port's background colour or pattern.

Main Constants, Data Types, and Functions

In the following, those constants and functions supported only under Mac OS 8.6 and later appear on a light gray background.

Constants

Window Class

```
KAlertWindowClass                = 1L
kMovableAlertWindowClass         = 2L
kModalWindowClass                = 3L
kMovableModalWindowClass         = 4L
kFloatingWindowClass             = 5L
kDocumentWindowClass             = 6L
```

Window Attributes

```
kWindowNoAttributes              = 0L
kWindowCloseBoxAttribute         = 1L << 0
kWindowHorizontalZoomAttribute   = 1L << 1
kWindowVerticalZoomAttribute     = 1L << 2
kWindowFullZoomAttribute         = kWindowVerticalZoomAttribute |
                                   kWindowHorizontalZoomAttribute
kWindowCollapseBoxAttribute      = 1L << 3
kWindowResizableAttribute        = 1L << 4
kWindowSideTitlebarAttribute     = 1L << 5
kWindowNoUpdatesAttribute        = 1L << 16
kWindowNoActivatesAttribute      = 1L << 17
kWindowStandardDocumentAttributes = kWindowCloseBoxAttribute |
                                   kWindowFullZoomAttribute |
                                   kWindowCollapseBoxAttribute |
                                   kWindowResizableAttribute)
kWindowStandardFloatingAttributes = kWindowCloseBoxAttribute |
                                   kWindowCollapseBoxAttribute
```

Window Positioning

```
kWindowCenterOnMainScreen        = 0x00000001
```

```
kWindowCenterOnParentWindow                = 0x00000002
kWindowCenterOnParentWindowScreen          = 0x00000003
kWindowCascadeOnMainScreen                 = 0x00000004
kWindowCascadeOnParentWindow               = 0x00000005
kWIndowCascadeOnParentWindowScreen         = 0x00000006
kWindowAlertPositionOnMainScreen           = 0x00000007
kWindowAlertPositionOnParentWindow         = 0x00000008
kWindowAlertPositionOnParentWindowScreen   = 0x00000009
```

Window Transition Action and Effect

```
kWindowShowTransitionAction                = 1
kWindowHideTransitionAction                = 2
kWindowZoomTransitionEffect                = 1
```

Data Types

Property Types

```
typedef OSType PropertyCreator;
typedef OSType PropertyTag;
```

Window Class and Attributes

```
typedef UInt32 WindowClass;
typedef UInt32 WindowAttributes;
```

Window Positioning

```
typedef UInt32 WindowPositionMethod;
```

Window Transitioning

```
typedef UInt32 WindowTransitionEffect;
typedef UInt32 WindowTransitionAction;
```

Functions

Floating Windows

```
OSStatus    InitFloatingWindows    (void);
OSStatus    HideFloatingWindows(void);
OSStatus    ShowFloatingWindows(void);
Boolean     AreFloatingWindowsVisible(void);
```

Window Proxy Icons

```
OSStatus    SetWindowProxyCreatorAndType(WindowPtr window,OSType fileCreator,OSType fileType,
            SInt16 vRefNum);
OSStatus    SetWindowProxyFSSpec(WindowPtr window,const FSSpec *inFile);
OSStatus    GetWindowProxyFSSpec(WindowPtr window,FSSpec * outFile);
OSStatus    GetWindowProxyAlias(WindowPtr window,AliasHandle *alias);
OSStatus    SetWindowProxyAlias(WindowPtr window,AliasHandle alias);
OSStatus    SetWindowProxyIcon(WindowPtr window,IconRef icon);
OSStatus    GetWindowProxyIcon(WindowPtr window,IconRef * outIcon);
OSStatus    RemoveWindowProxy(WindowPtr window);
OSStatus    TrackWindowProxyDrag(WindowPtr window,Point startPt);
```

Window Path Pop-Up Menus

```
Boolean     IsWindowPathSelectClick(WindowPtr window,EventRecord *event);
OSStatus    WindowPathSelect(WindowPtr    window,MenuHandle menu,SInt32 *outMenuResult);
```

Transitional Window Animations and Sounds

```
OSStatus    TransitionWindow(WindowPtr window,WindowTransitionEffect effect,
            WindowTransitionAction action,const Rect *rect);
```

Creating and Storing Windows

```
OSStatus      CreateNewWindow(WindowClass windowClass,WindowAttributes attributes,
              const Rect *bounds,WindowPtr *outWindow);
OSStatus      CreateWindowFromResource(SInt16 resID,WindowPtr *outWindow);
OSStatus      CreateWindowFromCollection(Collectioncollection,WindowPtr *outWindow);
OSStatus      StoreWindowIntoCollection(WindowPtr window,Collection collection);
```

Accessing Window Information

```
OSStatus      GetWindowClass(WindowPtr   window,WindowClass *outClass);
OSStatus      GetWindowAttributes  (WindowPtr window,WindowAttributes *outAttributes);
Boolean       IsValidWindowPtr(GrafPtr    grafPort);
WindowPtr     FrontNonFloatingWindow(void);
```

Zooming, Moving, Resizing, and Positioning Windows

```
OSStatus      ZoomWindowIdeal(WindowPtr window,SInt16 partCode,Point *ioIdealSize);
Boolean       IsWindowInStandardState(WindowPtr window,Point *idealSize,Rect *idealStandardState);
OSStatus      SetWindowIdealUserState(WindowPtr window,Rect *userState);
OSStatus      GetWindowIdealUserState(WindowPtr window,Rect *userState);
OSStatus      MoveWindowStructure(WindowPtr window,short hGlobal,short vGlobal);
OSStatus      SetWindowBounds(WindowPtr window,WindowRegionCode regionCode,
              const Rect *globalBounds);
OSStatus      GetWindowBounds(WindowPtr window,WindowRegionCode regionCode,Rect *globalBounds);
Boolean       ResizeWindow(WindowPtr window,Point startPoint,const Rect *sizeConstraints,
              Rect *newContentRect);
OSStatus      RepositionWindow(WindowPtr window,WindowPtr parentWindow,
              WindowPositionMethod method);
```

Associating Data With Windows

```
OSStatus      SetWindowProperty(WindowPtr window,PropertyCreator propertyCreator,
              PropertyTag propertyTag,UInt32 propertySize,void *propertyBuffer);
OSStatus      GetWindowProperty(WindowPtr window,PropertyCreator propertyCreator,
              PropertyTag propertyTag,UInt32 bufferSize,UInt32 *actualSize,void *propertyBuffer);
OSStatus      GetWindowPropertySize(WindowPtr window,PropertyCreator creator,PropertyTag tag,
              UInt32 *size);
OSStatus      RemoveWindowProperty(WindowPtr window,PropertyCreator propertyCreator,
              PropertyTag propertyTag);
```

Adding To and Removing From the Update Region

```
OSStatus      InvalWindowRect(WindowPtr window,const Rect *bounds);
OSStatus      InvalWindowRgn(WindowPtr window,RgnHandle region);
OSStatus      ValidWindowRect(WindowPtr window,const Rect *bounds);
OSStatus      ValidWindowRgn(WindowPtr window,RgnHandle region);
```

Setting Content Region Colour and Pattern

```
OSStatus      SetWindowContentColor(WindowPtr window,RGBColor *color);
OSStatus      GetWindowContentColor(WindowPtr window,RGBColor *color);
OSStatus      GetWindowContentPattern(WindowPtr window,PixPatHandle outPixPat);
OSStatus      SetWindowContentPattern(WindowPtr window,PixPatHandle pixPat);
```

Demonstration Program

```
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
// Windows2.c
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
//
// This program demonstrates Mac OS 8.5 Window Manager features and functions relating
// to:
//
// •   Creating floating windows and document windows using CreateNewWindow.  (Note that
//     floating windows will not be created unless the program is run under Mac OS 8.6 or
//     later.)
//
// •   Saving document windows and their associated data to a 'wind' resource.
//
// •   Creating document windows from 'wind' resources using CreateWindowFromResource.
//
// •   Managing windows in a floating windows environment.
//
// •   Setting and getting a window's property.
```

```
//
// •   Re-sizing a window using ResizeWindow and zooming a window using ZoomWindowIdeal.
//
// •   Showing and hiding windows using TransitionWindow.
//
// •   Displaying window proxy icons.
//
// Those aspects of the Mac OS 8.5 Window Manager not demonstrated in this program
// (full implementation of window proxy icons and window path pop-up menus) are
// demonstrated at the demonstration program associated with Chapter 16B (Files2).
//
// The program utilises the following resources:
//
// •   An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit, Document Windows
//     and Floating Windows menus (preload, non-purgeable).
//
// •   'TEXT' resources for the document windows (non-purgeable).
//
// •   'PICT' resources for the floating windows (non-purgeable).
//
// •   An 'ALRT' resource (purgeable), plus associated 'DITL', 'alrx', and 'dftb'
//     resources (all purgeable), for a modal alert box invoked when the user chooses the
//     About Windows2... item from the Apple menu.
//
// •   A 'SIZE' resource with the acceptSuspendResumeEvents, doesActivateOnFGSwitch,
//     and is32BitCompatible flags set.
//
// In addition, the program itself creates a 'wind' resource, and saves it to the
// resource fork of the application file, when the user chooses CreateNewWindow from the
// Document Windows menu.
//
// Note that floating windows will not be created unless the program is run under Mac OS
// 8.6 or later.
//
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

//
..............................................................................................................................................................
......................................... includes

#include <Appearance.h>
#include <Devices.h>
#include <Folders.h>
#include <Gestalt.h>
#include <Resources.h>
#include <Sound.h>
#include <ToolUtils.h>

//
..............................................................................................................................................................
............................................ defines

#define mApple                      128
#define  iAbout                     1
#define mFile                       129
#define mEdit                       130
#define  iQuit                      11
#define mDocumentWindows            131
#define  iCreateWindow              1
#define  iCreateFromResource 2
#define mFloatingWindows            132
#define  iColours                   1
#define  iTools                     2
#define rMenubar                    128
#define rAboutAlert                 128
#define rText                       128
#define rColoursPicture             128
#define rToolsPicture               129
#define rWind                       128
#define MIN(a,b)                    ((a) < (b) ? (a) : (b))

//
..............................................................................................................................................................
......................................... typedefs

typedef struct
{
   TEHandle  editStrucHdl;
} docStructure, **docStructureHandle;
```

```
// ..............................................................................................................................................................................
................. global variables

Boolean        gMacOS86Present    = false;
SInt16      gAppResFileRefNum;
WindowPtr   gColoursFloatingWindowPtr;
WindowPtr   gToolsFloatingWindowPtr;
Boolean        gDone;
Boolean        gInBackground;

// ..............................................................................................................................................................................
......... function prototypes

void  main                                   (void);
void  doInitManagers                       (void);
void  doEvents                               (EventRecord *);
void  doMouseDown                            (EventRecord *);
void  doUpdate                               (EventRecord *);
void  doUpdateDocumentWindow         (WindowPtr);
void  doActivate                             (EventRecord *);
void  doActivateDocumentWindow       (WindowPtr,Boolean);
void  doOSEvent                              (EventRecord *);
void  doAdjustMenus                        (void);
void  doMenuChoice                         (SInt32);
void  doDocumentWindowsMenu            (SInt16);
void  doFloatingWindowsMenu            (SInt16);
OSErr   doCreateNewWindow                      (void);
OSErr   doSaveWindow                          (WindowPtr);
OSErr   doCreateWindowFromResource    (void);
OSErr   doCreateFloatingWindows           (void);
void  doCloseWindow                          (WindowPtr);
void  doErrorAlert                        (SInt16);
void  doConcatPStrings                  (Str255,Str255);

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ main

void  main(void)
{
    OSErr                             osError;
    SInt32                        response;
    Handle                        menubarHdl;
    MenuHandle                menuHdl;
    EventRecord               eventStructure;
    SInt32                        sleepTime;
    WindowPtr                     windowPtr;
    docStructureHandle  docStrucHdl;
    UInt32                        actualSize;

    // ......................................................................................... check whether Mac OS 8.5 and 8.6 or later are present

    osError = Gestalt(gestaltSystemVersion,&response);

    if(osError == noErr && response < 0x00000850)
        ExitToShell();
    if(osError == noErr && response >= 0x00000860)
        gMacOS86Present = true;

    //
..............................................................................................................................................................................
... initialise managers

    doInitManagers();

    // ....................................................................................................................................................... set
up menu bar and menus

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

    menuHdl = GetMenuHandle(mApple);
    if(menuHdl == NULL)
        ExitToShell();
    else
```

```
        AppendResMenu(menuHdl,'DRVR');

    // .............................................................................. set current resource file to application resource fork

    gAppResFileRefNum = CurResFile();

    // ............................................................................................................................................................
create floating windows

    if(gMacOS86Present)
    {
        if(osError = doCreateFloatingWindows())
            doErrorAlert(osError);
    }
    else
    {
        menuHdl = GetMenuHandle(mFloatingWindows);
        DisableItem(menuHdl,0);
        menuHdl = GetMenuHandle(mApple);
        DisableItem(menuHdl,iAbout);
        DrawMenuBar();
    }

    //
..........................................................................................................................................................................
.............. enter eventLoop

    gDone = false;
    sleepTime = LMGetCaretTime();

    while(!gDone)
    {
        if(WaitNextEvent(everyEvent,&eventStructure,sleepTime,NULL))
            doEvents(&eventStructure);
        else
        {
            if(windowPtr = FrontNonFloatingWindow())
            {
                if(!(GetWindowProperty(windowPtr,0,'docs',sizeof(docStrucHdl),&actualSize,
                                                    &docStrucHdl)))
                    TEIdle((*docStrucHdl)->editStrucHdl);
            }
        }
    }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doInitManagers

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

    InitGraf(&qd.thePort);
    InitFonts();
    if(gMacOS86Present)
        InitFloatingWindows();
    else
        InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(NULL);

    InitCursor();
    FlushEvents(everyEvent,0);

    RegisterAppearanceClient();
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
    SInt8charCode;

    switch(eventStrucPtr->what)
    {
        case mouseDown:
```

```
      doMouseDown(eventStrucPtr);
      break;

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

    case updateEvt:
      doUpdate(eventStrucPtr);
      break;

    case activateEvt:
      doActivate(eventStrucPtr);
      break;

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

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doMouseDown

void doMouseDown(EventRecord *eventStrucPtr)
{
  WindowPtr     windowPtr;
  SInt16        partCode, zoomPart;
  WindowClass windowClass;
  Rect          growRect, newContentRect;
  SInt32        newSize;
  Point         idealSize;

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

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

    case inContent:
      GetWindowClass(windowPtr,&windowClass);
      if(windowClass == kFloatingWindowClass)
      {
        if(windowPtr != FrontWindow())
          SelectWindow(windowPtr);
        else
        {
          if(windowPtr == gColoursFloatingWindowPtr)
            ; // Appropriate action for Colours floating window here.
          else if(windowPtr == gToolsFloatingWindowPtr)
            ; // Appropriate action for Tools floating window here.
        }
      }
      else
      {
        if(windowPtr != FrontNonFloatingWindow())
          SelectWindow(windowPtr);
        else
          ; // Appropriate action for active document window here.
      }
      break;

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

    case inGoAway:
      GetWindowClass(windowPtr,&windowClass);
      if(windowClass == kFloatingWindowClass)
      {
```

```
        if(TrackGoAway(windowPtr,eventStrucPtr->where) == true)
          TransitionWindow(windowPtr,kWindowZoomTransitionEffect,
                           kWindowHideTransitionAction,NULL);
      }
      else
        if(TrackGoAway(windowPtr,eventStrucPtr->where) == true)
          doCloseWindow(windowPtr);
      break;

    case inGrow:
      growRect = qd.screenBits.bounds;
      growRect.top    = 80;
      growRect.left = 160;

      if(gMacOS86Present)
      {
        ResizeWindow(windowPtr,eventStrucPtr->where,&growRect,&newContentRect);
        InvalWindowRect(windowPtr,&windowPtr->portRect);
      }
      else
      {
        newSize = GrowWindow(windowPtr,eventStrucPtr->where,&growRect);
        if(newSize != 0)
        {
          SizeWindow(windowPtr,LoWord(newSize),HiWord(newSize),true);
          InvalWindowRect(windowPtr,&windowPtr->portRect);
        }
      }
      break;

    case inZoomIn:
    case inZoomOut:
      idealSize.v = qd.screenBits.bounds.bottom - 100;
      idealSize.h = qd.screenBits.bounds.right - 200;

      if(IsWindowInStandardState(windowPtr,&idealSize,NULL))
        zoomPart = inZoomIn;
      else
        zoomPart = inZoomOut;

      if(TrackBox(windowPtr,eventStrucPtr->where,partCode))
        ZoomWindowIdeal(windowPtr,zoomPart,&idealSize);
      break;
  }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doUpdate

void  doUpdate(EventRecord *eventStrucPtr)
{
  GrafPtr     oldPort;
  WindowPtr    windowPtr;

  GetPort(&oldPort);
  windowPtr = (WindowPtr) eventStrucPtr->message;

  BeginUpdate(windowPtr);

  SetPort(windowPtr);
  doUpdateDocumentWindow(windowPtr);

  EndUpdate(windowPtr);

  SetPort(oldPort);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doUpdateDocumentWindow

void  doUpdateDocumentWindow(WindowPtr windowPtr)
{
  Rect                  contentRect;
  OSStatus              osError;
  UInt32                actualSize;
  docStructureHandle  docStrucHdl;
  TEHandle              editStrucHdl;

  EraseRgn(windowPtr->visRgn);
  DrawGrowIcon(windowPtr);
```

```
    if(!(osError = GetWindowProperty(windowPtr,0,'docs',sizeof(docStrucHdl),&actualSize,
                                                    &docStrucHdl)))
    {
        contentRect = windowPtr->portRect;
        contentRect.right -= 15;
        contentRect.bottom -= 15;
        editStrucHdl = (*docStrucHdl)->editStrucHdl;
        (*editStrucHdl)->destRect = (*editStrucHdl)->viewRect = contentRect;
        TECalText(editStrucHdl);
        TEUpdate(&contentRect,(*docStrucHdl)->editStrucHdl);
    }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doActivate

void  doActivate(EventRecord *eventStrucPtr)
{
    WindowPtr    windowPtr;
    Boolean      becomingActive;

    windowPtr = (WindowPtr) eventStrucPtr->message;
    becomingActive = ((eventStrucPtr->modifiers & activeFlag) == activeFlag);

    doActivateDocumentWindow(windowPtr,becomingActive);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doActivateDocumentWindow

void  doActivateDocumentWindow(WindowPtr windowPtr,Boolean becomingActive)
{
    docStructureHandle  docStrucHdl;
    UInt32              actualSize;
    OSStatus            osError;

    if(!(osError = GetWindowProperty(windowPtr,0,'docs',sizeof(docStrucHdl),&actualSize,
                                                    &docStrucHdl)))
    {
        if(becomingActive)
            TEActivate((*docStrucHdl)->editStrucHdl);
        else
            TEDeactivate((*docStrucHdl)->editStrucHdl);
    }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doOSEvent

void  doOSEvent(EventRecord *eventStrucPtr)
{
    WindowPtr windowPtr;

    switch((eventStrucPtr->message >> 24) & 0x000000FF)
    {
        case suspendResumeMessage:
            gInBackground = (eventStrucPtr->message & resumeFlag) == 0;

            windowPtr = FrontNonFloatingWindow();
            if(windowPtr != NULL)
                doActivateDocumentWindow(windowPtr,!gInBackground);

            if(gInBackground)
                HideFloatingWindows();
            else
                ShowFloatingWindows();

            break;
    }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doAdjustMenus

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

    floatMenuHdl = GetMenuHandle(mFloatingWindows);

    isVisible = ((WindowPeek) gColoursFloatingWindowPtr)->visible;
    CheckItem(floatMenuHdl,iColours,isVisible);
```

```
    isVisible = ((WindowPeek) gToolsFloatingWindowPtr)->visible;
    CheckItem(floatMenuHdl,iTools,isVisible);

    DrawMenuBar();
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doMenuChoice

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

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

    if(menuID == 0)
        return;

    switch(menuID)
    {
        case mApple:
            if(menuItem == iAbout)
            {
                Alert(rAboutAlert,NULL);
                HiliteWindow(gColoursFloatingWindowPtr,true);
                HiliteWindow(gToolsFloatingWindowPtr,true);
            }
            else
            {
                GetMenuItemText(GetMenuHandle(mApple),menuItem,itemName);
                daDriverRefNum = OpenDeskAcc(itemName);
            }
            break;

        case mFile:
            if(menuItem == iQuit)
                gDone = true;
            break;

        case mDocumentWindows:
            doDocumentWindowsMenu(menuItem);
            break;

        case mFloatingWindows:
            doFloatingWindowsMenu(menuItem);
            break;
    }

    HiliteMenu(0);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doDocumentWindowsMenu

void  doDocumentWindowsMenu(SInt16 menuItem)
{
    OSErr   osError;

    switch(menuItem)
    {
        case iCreateWindow:
            if(osError = doCreateNewWindow())
                doErrorAlert(osError);
            break;

        case iCreateFromResource:
            if(osError = doCreateWindowFromResource())
                doErrorAlert(osError);
            break;
    }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doFloatingWindowsMenu

void  doFloatingWindowsMenu(SInt16 menuItem)
{
    Boolean    isVisible;
```

```
    if(menuItem == iColours)
    {
        isVisible = ((WindowPeek) gColoursFloatingWindowPtr)->visible;
        if(isVisible)
            TransitionWindow(gColoursFloatingWindowPtr,kWindowZoomTransitionEffect,
                                kWindowHideTransitionAction,NULL);
        else
            TransitionWindow(gColoursFloatingWindowPtr,kWindowZoomTransitionEffect,
                                kWindowShowTransitionAction,NULL);
    }
    else if(menuItem == iTools)
    {
        isVisible = ((WindowPeek) gToolsFloatingWindowPtr)->visible;
        if(isVisible)
            TransitionWindow(gToolsFloatingWindowPtr,kWindowZoomTransitionEffect,
                                kWindowHideTransitionAction,NULL);
        else
            TransitionWindow(gToolsFloatingWindowPtr,kWindowZoomTransitionEffect,
                                kWindowShowTransitionAction,NULL);
    }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doCreateFloatingWindows

OSErr  doCreateFloatingWindows(void)
{
    Rect        contentRect;
    OSStatus    osError;
    PicHandle pictureHdl;

    SetRect(&contentRect,102,59,391,132);

    if(!(osError = CreateNewWindow(kFloatingWindowClass,
                                            kWindowStandardFloatingAttributes +
                                            kWindowSideTitlebarAttribute,
                                            &contentRect,&gColoursFloatingWindowPtr)))
    {
        if(pictureHdl = GetPicture(rColoursPicture))
            SetWindowPic(gColoursFloatingWindowPtr,pictureHdl);
        osError = TransitionWindow(gColoursFloatingWindowPtr,kWindowZoomTransitionEffect,
                                            kWindowShowTransitionAction,NULL);
    }

    if(osError != noErr)
        return osError;

    SetRect(&contentRect,149,88,213,280);

    if(!(osError = CreateNewWindow(kFloatingWindowClass,
                                            kWindowStandardFloatingAttributes,
                                            &contentRect,&gToolsFloatingWindowPtr)))
    {
        if(pictureHdl = GetPicture(rToolsPicture))
            SetWindowPic(gToolsFloatingWindowPtr,pictureHdl);

        osError = TransitionWindow(gToolsFloatingWindowPtr,kWindowZoomTransitionEffect,
                                            kWindowShowTransitionAction,NULL);
    }

    return osError;
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doCreateNewWindow

OSErr  doCreateNewWindow(void)
{
    Rect                    contentRect;
    OSStatus                osError;
    WindowPtr                   windowPtr;
    docStructureHandle  docStrucHdl;
    Handle                  textHdl;

    SetRect(&contentRect,10,40,470,340);

    do
    {
        if(osError = CreateNewWindow(kDocumentWindowClass,kWindowStandardDocumentAttributes,
```

```
                                                  &contentRect,&windowPtr))
      break;

    if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
    {
      osError = MemError();
      break;
    }

    if(osError = SetWindowProperty(windowPtr,0,'docs',sizeof(docStructure),
                                                    &docStrucHdl))
      break;

    SetPort(windowPtr);
    TextSize(10);

    textHdl = GetResource('TEXT',rText);
    osError = ResError();
    if(osError != noErr)
      break;

    OffsetRect(&contentRect,-contentRect.left,-contentRect.top);
    contentRect.right -= 15;
    contentRect.bottom -= 15;

    (*docStrucHdl)->editStrucHdl = TENew(&contentRect,&contentRect);
    TEInsert(*textHdl,GetHandleSize(textHdl),(*docStrucHdl)->editStrucHdl);

    SetWTitle(windowPtr,"\pCreateNewWindow");

    if(osError = SetWindowProxyCreatorAndType(windowPtr,0,'TEXT',kOnSystemDisk))
      break;
    if(osError = SetWindowModified(windowPtr,false))
      break;
    if(osError = RepositionWindow(windowPtr,NULL,kWindowCascadeOnMainScreen))
      break;
    if(osError = TransitionWindow(windowPtr,kWindowZoomTransitionEffect,
                                                    kWindowShowTransitionAction,NULL))
      break;

    if(osError = doSaveWindow(windowPtr))
      break;

  } while(false);

  if(osError)
  {
    if(windowPtr)
      DisposeWindow(windowPtr);

    if(docStrucHdl)
      DisposeHandle((Handle) docStrucHdl);
  }

  return osError;
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doSaveWindow

OSErr  doSaveWindow(WindowPtr windowPtr)
{
  Collection          collection = NULL;
  OSStatus            osError;
  docStructureHandle  docStrucHdl;
  UInt32              actualSize;
  Handle              flatCollectHdl, flatCollectResHdl, existingResHdl;

  do
  {
    if(!(collection = NewCollection()))
    {
      osError = MemError();
      break;
    }

    if(osError = StoreWindowIntoCollection(windowPtr,collection))
      break;

    if(osError = GetWindowProperty(windowPtr,0,'docs',sizeof(docStrucHdl),&actualSize,
```

```
                                                &docStrucHdl))
    break;

if(osError = AddCollectionItemHdl(collection,'TEXT',1,
                                                        (*(*docStrucHdl)->editStrucHdl)->hText))
    break;

if(!(flatCollectHdl = NewHandle(0)))
{
    osError = MemError();
    break;
}

if(osError = FlattenCollectionToHdl(collection,flatCollectHdl))
    break;

existingResHdl = Get1Resource('wind',rWind);
osError = ResError();
if(osError != noErr && osError != resNotFound)
    break;

if(existingResHdl != NULL)
    RemoveResource(existingResHdl);
osError = ResError();
if(osError != noErr)
    break;

AddResource(flatCollectHdl,'wind',rWind,"\p");
osError = ResError();
if(osError != noErr)
    break;

flatCollectResHdl = flatCollectHdl;
flatCollectHdl = NULL;

WriteResource(flatCollectResHdl);
osError = ResError();
if(osError != noErr)
    break;

        UpdateResFile(gAppResFileRefNum);
        osError = ResError();
        if(osError != noErr)
            break;
    } while(false);

    if(collection)
        DisposeCollection(collection);
    if(flatCollectHdl)
        DisposeHandle(flatCollectHdl);
    if(flatCollectResHdl)
        ReleaseResource(flatCollectResHdl);

    return osError;
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doCreateWindowFromResource

OSErr  doCreateWindowFromResource(void)
{
    OSStatus                osError;
    WindowPtr                   windowPtr;
    Collection              unflattenedCollection = NULL;
    Handle                  windResHdl;
    docStructureHandle  docStrucHdl;
    SInt32                  dataSize = 0;
    Handle                  textHdl;
    Rect                    contentRect;

    do
    {
        if(osError = CreateWindowFromResource(rWind,&windowPtr))
            break;

        if(!(unflattenedCollection = NewCollection()))
        {
            osError = MemError();
            break;
        }
```

```
    windResHdl = GetResource('wind',rWind);
    osError = ResError();
    if(osError != noErr)
      break;

    if(osError = UnflattenCollectionFromHdl(unflattenedCollection,windResHdl))
      break;

    if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
    {
      osError = MemError();
      break;
    }

    if(osError = GetCollectionItem(unflattenedCollection,'TEXT',1,&dataSize,
                                   kCollectionDontWantData))
      break;

    if(!(textHdl = NewHandle(dataSize)))
    {
      osError = MemError();
      break;
    }

    if(osError = GetCollectionItem(unflattenedCollection,'TEXT',1,kCollectionDontWantSize,
                                   *textHdl))
      break;

    contentRect = windowPtr->portRect;
    contentRect.right -= 15;
    contentRect.bottom -= 15;
    SetPort(windowPtr);
    TextSize(10);
    (*docStrucHdl)->editStrucHdl = TENew(&contentRect,&contentRect);
    TEInsert(*textHdl,dataSize,(*docStrucHdl)->editStrucHdl);

    if(osError = SetWindowProperty(windowPtr,0,'docs',sizeof(docStrucHdl),&docStrucHdl))
      break;

    SetWTitle(windowPtr,"\pCreateWindowFromResource");

    if(osError = SetWindowProxyCreatorAndType(windowPtr,0,'TEXT',kOnSystemDisk))
      break;
    if(osError = SetWindowModified(windowPtr,false))
      break;
    if(osError = RepositionWindow(windowPtr,NULL,kWindowCascadeOnMainScreen))
      break;
    if(osError = TransitionWindow(windowPtr,kWindowZoomTransitionEffect,
                                  kWindowShowTransitionAction,NULL))
      break;
  } while(false);

  if(unflattenedCollection)
    DisposeCollection(unflattenedCollection);
  if(windResHdl)
    ReleaseResource(windResHdl);

  return osError;
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doCloseWindow

void  doCloseWindow(WindowPtr windowPtr)
{
  OSStatus            osError;
  docStructureHandle  docStrucHdl;
  UInt32              actualSize;

  do
  {
    if(osError = TransitionWindow(windowPtr,kWindowZoomTransitionEffect,
                                  kWindowHideTransitionAction,NULL))
    break;

    if(osError = GetWindowProperty(windowPtr,0,'docs',sizeof(docStrucHdl),&actualSize,
                                   &docStrucHdl))
    break;
  } while(false);
```

```
    if(osError)
        doErrorAlert(osError);

    if((*docStrucHdl)->editStrucHdl)
        TEDispose((*docStrucHdl)->editStrucHdl);

    if(docStrucHdl)
        DisposeHandle((Handle) docStrucHdl);

    DisposeWindow(windowPtr);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doErrorAlert

void  doErrorAlert(SInt16 errorCode)
{
    AlertStdAlertParamRec paramRec;
    Str255                errorCodeString;
    Str255                theString = "\pAn error occurred.  The error code is ";
    SInt16                itemHit;

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

    NumToString((SInt32) errorCode,errorCodeString);
    doConcatPStrings(theString,errorCodeString);

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

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doConcatPStrings

void  doConcatPStrings(Str255 targetString,Str255 appendString)
{
    SInt16  appendLength;

    appendLength = MIN(appendString[0],255 - targetString[0]);

    if(appendLength > 0)
    {
        BlockMoveData(appendString+1,targetString+targetString[0]+1,(SInt32) appendLength);
        targetString[0] += appendLength;
    }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
```

Demonstration Program Comments

This program will run only under Mac OS 8.5 or later. The program's floating windows will be created only if the program is run under Mac OS 8.6 or later.

Two Mac OS 8.5 Window Manager features (full window proxy icon implementation and window path pop-up menus) are not demonstrated in this program. However, they are demonstrated at the demonstration program associated with Chapter 16B (Files2).

When the program is run, the user should:

- Choose CreateNewWindow from the Document Windows menu, noting that, when the new window is displayed, the floating windows and the new (document) window are all active.

  (Note: As well as creating the window, the program loads and displays a 'TEXT' resource (simulating a document associated with the window) and then saves the window and the text to a 'wind' resource.)

- Choose CreateWindowFromResource from the Document Windows menu, noting that the window is created from the 'wind' resource saved when CreateNewWindow was chosen.

- Choose About Windows2... from the Apple menu, noting that the floating windows appear in the deactivated state when the alert box opens.
- Hide the floating windows by clicking their close boxes, and toggle the floating windows between hidden and showing by choosing their items in the Floating Windows menu, noting the transitional animations and sounds.
- Click in the Finder to send the application to the background, noting that the floating windows are hidden by this action. Then click in one of the application's windows, noting that the floating windows re-appear.
- Drag a document window to various locations on the screen, zoom the window in and out at those locations, and note that the zoom out to the standard state conforms with human interface guidelines.
- Resize the document windows.
- Note the transitional animations and sounds when the document windows are opened and closed.

#define

The first twelve #defines establish constants representing menu IDs and resources, menu item numbers, and the menu bar resource ID. The next five represent the resource IDs for an 'ALRT' resource (and associated 'DITL', 'alrx', and 'dfnt' resources), a 'TEXT' resource, two 'PICT' resources, and the 'wind' resource created by the program. The (fairly common) macro which follows is required by the application-defined string concatenation doConcatPStrings.

#typedef

A document structure of type docStructure will be "attached" to each document window. The single field in the document structure (editStrucHdl) will be assigned a handle to a TextEdit edit structure, which will contain the text displayed in the window.

Global Variables

gMacOS86Present will be assigned true if Mac OS 8.6 or later is present. gAppResFileRefNum will be assigned the file reference number for the application file's resource fork. gColoursFloatingWindowPtr and gToolsFloatingWindowPtr will be assigned pointers to the colour graphics port structures for the floating windows. gDone, when set to true, will cause the main event loop to exit and the program to terminate. gInBackground relates to foreground/background switching.

main

Gestalt is called to determine which version of the Mac OS is present. If Mac OS 8.5 or later is not present, the program terminates. If Mac OS 8.6 or later is present, the global variable gMacOS86Present is set to true.

The next block sets up the menus. Note that error handling in this block is very rudimentary: the program simply terminates.

At the next block CurResFile is called to set the application's resource fork as the current resource file. This is necessary because the program will be saving a 'wind' resource to the application file's resource fork.

If Mac OS 8.6 is present, an application-defined function is called to create and show the floating windows, otherwise the Floating Windows menu and the About Windows2... item in the Apple menu are disabled.

In the next block (the main event loop), WaitNextEvent's sleep parameter is assigned the value returned by LMGetCaretTime. (LMGetCaretTime returns the value stored in the low memory global CaretTime, which determines the blinking rate for the insertion point caret as set by the user using the General Controls Control Panel.) This ensures that TEIdle, which causes the caret to blink, will be called at the correct interval.

When WaitNextEvent returns a NULL event, the Mac OS 8.5 function FrontNonFloatingWindow is called to obtain a pointer to the front document window. If such a window exists, the Mac OS 8.5 function GetWindowProperty is called to retrieve a handle to the window's document structure. The handle to the TextEdit edit structure, which is stored in the window's document structure, is then passed in the call to TEIdle, which causes the caret to blink.

doInitManagers

Note that, if Mac OS 8.6 is present, the Mac OS 8.5 function InitFloatingWindows is called, otherwise InitWindows is called.

doMouseDown

doMouseDown continues the processing of mouse-down events, switching according to the part code.

The inContent case is handled differently depending on whether the event is in a floating window or a document window. The Mac OS 8.5 function GetWindowClass returns the window's class. If the window is a floating window, and if that window is not the front floating window, SelectWindow is called to bring that floating window to the front. If the window is the front floating window, the identity of the window is determined and the appropriate further action is taken. (In this demonstration, no further action is taken.)

If the window is not a floating window, and if the window is not the front non-floating window, SelectWindow is called to:

- Unhighlight the currently active non-floating window, bring the specified window to the front of the non-floating windows, and highlight it.

- Generate activate events for the two windows.

- Move the previously active non-floating window to a position immediately behind the specified window.

If the window is the front non-floating window, the appropriate further action is taken. (In this demonstration, no further action is taken.)

The inGoAway case is also handled differently depending on whether the event is in a floating window or a document window. TrackGoAway is called in both cases to track user action while the mouse-button remains down. If the pointer is still within the go away box when the mouse-button is released, and if the window is a floating window, the Mac OS 8.5 function TransitionWindow is called to hide the window. If the window is a non-floating window, the application-defined function doCloseWindow is called to close the window.

The inGrow case first sets up the Rect passed in the third parameter of the ResizeWindow and GrowWindow calls which, in turn, will limit the minimum and maximum sizes to which the window can be resized. The top, left, bottom and right fields must contain, respectively, the minimum vertical, the minimum horizontal, the maximum vertical, and the maximum horizontal measurements. At the first line, this Rect is set to the boundaries of the screen, which is a reasonable way to get reasonable values into the bottom and right fields. The top and left fields, however, need to be manually set to some reasonable values (the two next lines).

If Mac OS 8.6 or later is present, the Mac OS 8.6 function ResizeWindow is called to retain control while the mouse-button remains down, and to draw the window frame in its new size when the mouse button is released. (When ResizeWindow returns, its newContentRect parameter contains the new dimension of the window's content region in global coordinates, although this information is not used in this demonstration.) The call to the Mac OS 8.5 function InvalWindowRect will cause the entire content region to be erased and redrawn (see doUpdateDocumentWindow).

If Mac OS 8.5 or earlier is present, GrowWindow is called and retains control until the user releases the mouse button, at which time the Rect variable newSize will contain the new window size coordinates. (Note that GrowWindow does not redraw the window in this size.) The SizeWindow call then redraws the window frame. The call to the Mac OS 8.5 function InvalWindowRect will cause the entire content region to be erased and redrawn (see doUpdateDocumentWindow).

At the inZoomIn and InZoomOut cases, the first two lines set the fields of a Point variable to equal the height and width of the main screen less 100 and 200 pixels respectively. This represents what this application considers to be the ideal size for the window. The call to the Mac OS 8.5 function IsWindowInStandardState compares the window's current size with this ideal size and, if they are equal, inZoomIn is assigned to the local variable zoomPart, meaning that the window is to be zoomed in to the user state. If they are not equal, inZoomOut is assigned to zoomPart, meaning that the window is to be zoomed out to the standard state. TrackBox retains control until the user releases the mouse button. If the pointer is still within the zoom box when the mouse button is released, the Mac OS 8.5 function ZoomWindowIdeal is called to zoom the window in accordance with human interface guidelines, and in the direction specified by zoomPart.

doUpdate

doUpdate further processes update events. When an update event is received, doUpdate calls doUpdateDocumentWindow. (As will be seen, in this particular demonstration, the Window Manager will not generate updates for the floating windows.)

doUpdateDocumentWindow

doUpdateDocumentWindow is concerned with the redrawing of the content region of the non-floating windows.

Firstly, the window's visible region, which at this point equates to the update region, is erased and DrawGrowIcon is called to draw the scroll bar delimiting lines. (This latter call is for cosmetic purposes only and would not be made if the window contained scroll bars.)

The Mac OS 8.5 function GetWindowProperty is then called to retrieve the handle to the window's document structure, which, as previously stated, contains a handle to a TextEdit edit text structure containing the text displayed in the window. If the call is successful, measures are taken to redraw the text in the window, taking account of the current height and width of the content region less the area that would ordinarily be occupied by scroll bars. (The TextEdit is calls in this section are incidental to the demonstration. TextEdit is addressed at Chapter 19 — Text and TextEdit.)

doActivate

doActivate attends to those aspects of window activation not handled by the Window Manager. The modifiers field of the event structure is tested to determine whether the window in question is being activated or deactivated. The result of this test is passed as a parameter in a call to the application-defined function for activating non-floating windows.

doActivateDocumentWindow

doActivateDocumentWindow performs, for the non-floating windows, those window activation actions for which the application is responsible. In this demonstration, that action is limited to calling TEActivate or TEDeactivate to show or remove the insertion point caret.

The Mac OS 8.5 function GetWindowProperty is called to retrieve the handle to the window's document structure, which contains a handle to the TextEdit edit text structure containing the text displayed in the window. If this call is successful, and if the window is being activated, TEActivate is called to display the insertion point caret. If the window is being deactivated, TEDeactivate is called to remove the insertion point caret.

doOSEvent

doOSEvent handles operating system events. In this demonstration, action is taken only in the case of suspend and resume messages.

If the event is a suspend event, the global variable gInBackground is assigned true, otherwise it is assigned false. The call to the Mac OS 8.5 function FrontNonFloatingWindow and the following line determines whether there are any non-floating windows present. If so, a pointer to the front non-floating window is passed in the call to the application-defined function for activating and deactivating no-floating windows. In the next block, if the event was a suspend event, the Mac OS 8.5 function HideFloatingWindows is called to hide the floating windows, otherwise the Mac OS 8.5 function ShowFloatingWindows is called to show the floating windows.

doAdjustMenus

doAdjustMenus is called in the event of a mouse-down event in the menu bar when a key is pressed together with the Command key. The function checks or unchecks the items in the Floating Windows menu depending on whether the associated floating window is currently showing or hidden.

doMenuChoice

doMenuChoice switches according to the menu choices of the user.

If the user chooses the About Windows2... item from the Apple menu, Alert is called to display the About Windows2... alert box.

The calls to HiliteWindow explicitly activate the two floating windows when the alert box is dismissed. This is a workaround to compensate for what appears to be a Window Manager window activation anomaly. This anomaly is evidenced as follows:

- When the application is launched and a document window is opened before the alert box is invoked, the floating windows appear in the deactivated state while the alert box is present and in the activated state when the alert box is dismissed. This is the expected correct behaviour.

- However, if a document window is not opened before the alert box is invoked, the floating windows remain in the deactivated state when the alert box is dismissed.

The calls to HiliteWindow have been included to cater for the second of the above situations.

If the user chooses the Quit item in the File menu, the global variable gDone is set to true, causing the program to terminate.

doDocumentWindowsMenu

doDocumentWindowsMenu further processes choices from the Document Windows menu. If the user chose the first item, the application-defined function doCreateNewWindow is called. If the user chose the second item, the application-defined function doCreateWindowFromResource is called. If either of these functions return an error, an application-defined error-handling function is called.

doFloatingWindowsMenu

doFloatingWindowsMenu further processes choices from the Floating Windows menu.

When an item is chosen, the visible field of the window's colour window structure is examined to determine whether the window is currently showing or hidden. The Mac OS 8.5 function TransitionWindow is then called, with the appropriate constant passed in the action parameter, to hide or show the window, depending on the previously determined current visibility state.

doCreateFloatingWindows

doCreateFloatingWindows is called from main to create the floating windows.

The Colours floating window is created first. SetRect is called to define a rectangle which will be used to establish the size of the window and its opening location in global coordinates. The Mac OS 8.5 function CreateNewWindow is then called to create a floating window (first parameter) with a close box, a collapse box, and a side title bar (second parameter), and with the previously defined content region size and location (third parameter).

If this call is successful, GetPicture is called to load the specified 'PICT' resource. If the resource is loaded successfully, SetWindowPic is called to store the handle to the picture structure in the windowPic field of the window's colour window structure. This latter means that the Window Manager will draw the picture in the window instead of generating update events for it. Finally, the Mac OS 8.5 function TransitionWindow is called to make the window visible (with animation and sound).

The same general procedure is then followed to create the Tools floating window.

doCreateNewWindow

doCreateNewWindow is called when the user chooses Create New Window from the Document Windows menu. In addition to creating a window, and for the purposes of this demonstration, doCreateNewWindow also saves the window and its associated data (text) in a 'wind' resource.

Firstly, SetRect is called to define a rectangle that will be used to establish the size of the window and its opening location in global coordinates. The call to the Mac OS 8.5 function CreateNewWindow creates a document window (first parameter) with a close box, a full zoom box, a collapse box, and a size box (second parameter), and with the previously defined content region size and location (third parameter).

NewHandle is then called to create a relocatable block for the document structure to be associated with the window. The Mac OS 8.5 function SetWindowProperty associates the document structure with the window. 0 is passed in the propertyCreator parameter because this demonstration has no application signature. The value passed in the propertyTag parameter ('docs') is just a convenient value with which to identify the data.

The call to SetPort sets the window's colour graphics port as the current port and the call to TextSize ensures that the size of the text to be drawn in the window will be 10 points.

The next three blocks load a 'TEXT' resource, insert the text into a TextEdit edit text structure, and assign a handle to that structure to the editStrucHdl field of the window's document structure. This is all for the purpose of simulating some text that the user has typed into the window.

SetWTitle sets the window's title.

The window lacks an associated file, so the Mac OS 8.5 function SetWindowProxyCreatorAndType is called to cause a proxy icon to be displayed in the window's drag bar. 0 passed in the fileCreator parameter and 'TEXT' passed in the fileType parameter cause the system's default icon for a document file to be displayed. The Mac OS 8.5 function SetWindowModified is then called with false passed in the modified parameter to cause the proxy icon to appear in the enabled state (indicating no unsaved changes).

The call to the Mac OS 8.5 function RepositionWindow positions the window relative to other windows according to the constant passed in the method parameter.

As the final step in creating the window, the Mac OS 8.5 function TransitionWindow is called to make the window visible (with animation and sound).

To facilitate the demonstration of creating a window from a 'wind' resource (see the function doCreateWindowFromResource), an application-defined function is called to save the window and its data (the text) to a 'wind' resource in the application's resource fork.

If an error occurred within the do/while loop, if a window was created, it is disposed of. Also, if a nonrelocatable block for the document structure was created, it is disposed of.

doSaveWindow

doSaveWindow is called by doCreateNewWindow to save the window and its data (the text) to a 'wind' resource.

The call to the Collection Manager function NewCollection allocates memory for as new collection object and initializes it. The call to the Mac OS 8.5 function StoreWindowIntoCollection stores data describing the window into the collection.

The Mac OS 8.5 function GetWindowProperty retrieves the handle to the window's document structure.

The handle to the window's text is is stored in the hText field of the TextEdit edit text structure. The handle to the edit text structure is, in turn, stored in the window's document structure. The Collection Manager function AddCollectionItemHdl adds a new item to the collection, specifically, a copy of the text.

The call to NewHandle allocates a zero-length handle which will be used to hold a flattened collection. The Collection Manager function FlattenCollectionToHdl flattens the collection into a Memory Manager handle.

The next six blocks use Resource Manager functions to save the flattened collection as a 'wind' resource in the resource fork of the application file.

Get1Resource attempts to load a 'wind' resource with ID 128. If ResError reports an error, and if the error is not the "resource not found" error, the whole save process is aborted. (Accepting the "resource not found" error as an acceptable error caters for the possibility that this may be the first time the window and its data have been saved.)

If Get1Resource successfully loaded a 'wind' resource with ID 128, RemoveResource is called to remove that resource from the resource map, AddResource is called to make the flattened collection in memory into a 'wind' resource, assigning a resource type, ID and name to that resource, and inserting an entry in the resource map for the current resource file. WriteResource is called to write the resource to the application's resource fork. Since the resource map has been changed, UpdateResFile is called to update the resource map on disk.

Below the do/while loop, the collection and the flattened collection block are disposed of and the resource in memory is released.

doCreateWindowFromResource

doCreateWindowFromResource creates a window from the 'wind' resource created by doSaveWindow.

The Mac OS 8.5 function CreateWindowFromResource creates a window, invisibly, from the 'wind' resource with ID 128.

The call to the Collection Manager function NewCollection creates a new collection. GetResource loads the 'wind' resource with ID 128. The Collection Manager function UnflattenCollectionFromHdl unflattens the 'wind' resource and stores the unflattened collection in the collection object unflattenedCollection.

NewHandle allocates a relocatable block the size of a window document structure.

The Collection Manager function GetCollectionItem is called twice, the first time to get the size of the text data, not the data itself. (The item in the collection is specified by the second and third parameters (tag and ID)). This allows the call to NewHandle to create a relocatable block of the same size. GetCollection is then called again, this time to obtain a copy of the text itself.

The next block creates a new TextEdit edit text structure (TENew), assigning its handle to the editStrucHdl field of the document structure which will shortly be associated with the window. TEInsert inserts the copy of the text obtained by the second call to GetCollectionItem into the edit text structure.

The call to the Mac OS 8.5 function SetWindowProperty associates the document structure with the window, thus associating the edit text structure and its text with the window.

SetWTitle sets the window's title.

The window lacks an associated file, so the Mac OS 8.5 function SetWindowProxyCreatorAndType is called to cause a proxy icon to be displayed in the window's drag bar. 0 passed in the fileCreator parameter and 'TEXT' passed in the fileType parameter cause the system's default icon for a document file to be displayed. The Mac OS 8.5 function SetWindowModified is then called with false passed in the modified parameter to cause the proxy icon to appear in the enabled state (indicating no unsaved changes).

The call to the Mac OS 8.5 function RepositionWindow positions the window relative to other windows according to the constant passed in the method parameter.

As the final step in creating the window, the Mac OS 8.5 function TransitionWindow is called to make the window visible (with animation and sound).

Below the do/while loop, the unflattened collection is disposed of and the 'wind' resource is released.

doCloseWindow

doCloseWindow is called when the user clicks the close box of a document window.

TransitionWindow is called to hide the window (with animation and sound). The Mac OS 8.5 function GetWindowProperty is then called to retrieve a handle to the window's document structure, allowing the memory occupied by the edit text structure and document structure associated with the window to be disposed of. DisposeWindow is then called to remove the window from the window list and discard all its data storage.

doErrorAlert and doConcatPStrings

doErrorAlert is called when errors are detected. In this demonstration, the action taken is somewhat rudimentary. A stop alert box displaying the error number is invoked. When the user dismisses the alert box, the program terminates.

doConcatPStrings is called from doErrorAlert to concatenate two strings into the single string displayed in the alert box.