16

MORE ON WINDOWS

Demonstration Program: Windows2

Introduction

As stated at Chapter 4, Mac OS 8.5 introduced a number of new features and associated system software functions to the Window Manager. The new functions associated with zooming and re-sizing windows were addressed at Chapter 4. This chapter addresses the remaining additional features, which include:

- Support for:
  - Floating windows.
  - Window proxy icons.
  - Window path pop-up menus.
  - Transitional window animations and sounds.
- New functions for:
  - Creating and storing windows.
  - Accessing window information.
  - Moving 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.

This chapter also addresses additional features introduced with Carbon and live window resizing introduced with Mac OS X.

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.

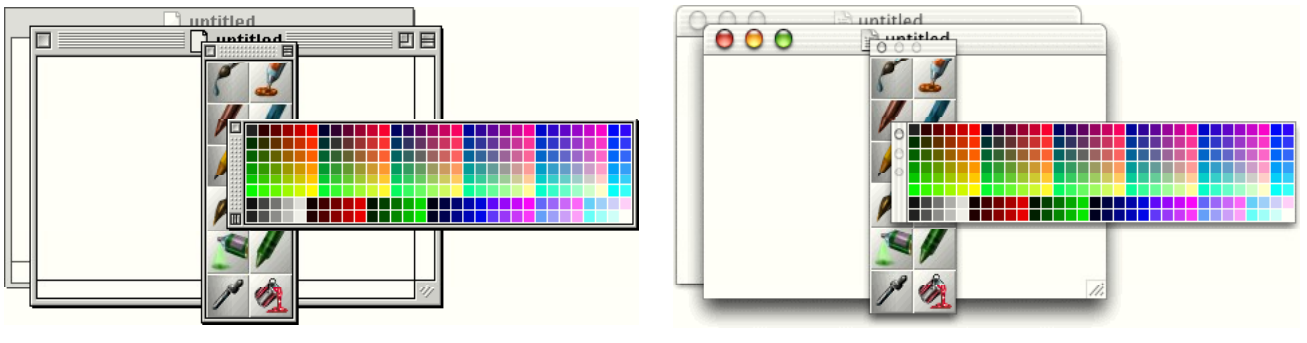

FIG 1 - FLOATING WINDOWS

In terms of front-to-back ordering of onscreen objects, 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 when a modal or movable modal dialog or alert is presented to the user. When this occurs, the appearance of all floating windows changes to reflect the inactive state.

Floating Window Types

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

Opening, Closing, Showing, and Hiding Floating Windows

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

When a floating window is created, it should remain open until the application is closed, and your application should provide the user with a means to hide or show the window as and when required. Ordinarily, it should do this by providing an item in an appropriate menu which allows the user to toggle the window between the hidden and showing states.

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

Floating windows should be hidden when the 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.

> ### Carbon Note
>
> In Carbon applications, floating windows are hidden and shown automatically on suspend and resume events. It is thus not necessary for Carbon applications to call HideFloatingWindows and ShowFloatingWindows.

Functions Relating to Floating Windows

The following function is relevant to floating windows:

Function	Description
AreFloatingWindowsVisible	Indicates whether an application's floating windows are visible.

Utility and Toolbar Windows

Carbon introduced the **utility window** (a system-wide floating window which floats above all other windows) and the **toolbar window**, which floats above all document windows in an application but below floating windows. (See Window Class Constants, below.)

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.

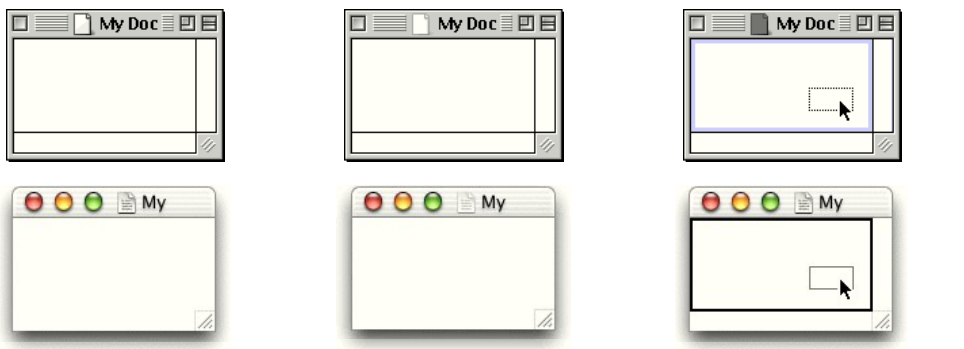

FIG 2 - WINDOW PROXY ICONS

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. 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. (On Mac OS X, this is accompanied by a dot appearing in the middle of the close button.) 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.

Proxy Icons and File Synchronisation Functions

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 are addressed at Chapter 18.

Functions Relating to Window Proxy Icons

The following functions are relevant to window proxy icons:

Function	Description
SetWindowProxyCreatorAndType	Sets the proxy icon for a window that has no associated file. New untitled windows should have a proxy icon so as to be consistent, in terms of appearance, with other windows.
SetWindowProxyFSSpec	Associates a file with a window using a file system specification (FSSpec) structure, thus establishing a proxy icon for the window.
GetWindowProxyFSSpec	Gets a file system specification (FSSpec) structure for the file 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	Gets alias data for the file associated with a window.
SetWindowProxyIcon	Overrides the default proxy icon for a window.
GetWindowProxyIcon	Gets a window's proxy icon.
RemoveWindowProxy	Dissociates a file from a window.
TrackWindowProxyDrag	Handles all aspects of the drag process when a proxy icon is dragged by the user.

Note that SetPortWindowPort (or SetPort) should be called to set the relevant window's 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.

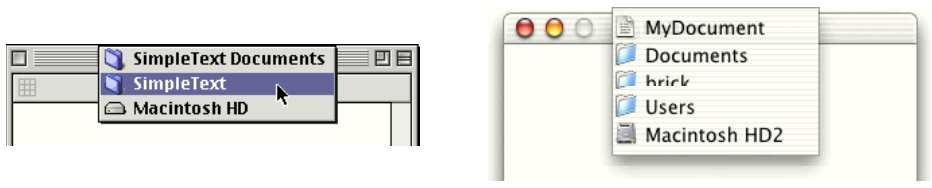

FIG 3 - WINDOW PATH POP-UP MENUS

Displaying and Handling a Window Path Pop-Up Menu

The proxy icon region overlays the title text region which, in turn, overlays the drag region (see Fig 4). Your application must be prepared to respond to a Command-click in either region.

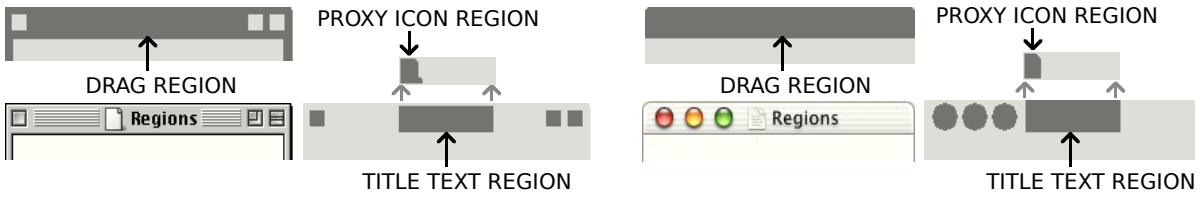

FIG 4 - DRAG, TITLE TEXT, AND PROXY ICON REGIONS

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 a function which makes the Finder the frontmost process.

Window path pop-up menus are demonstrated at the demonstration program associated with Chapter 18.

Transitional Window Animation and Sounds

On Mac OS 8/9, prior to Mac OS 8.5, the Window Manager supported the playing of a sound to accompany the animation that occured when a user clicked a window's collapse box. Mac OS 8.5 added support for animation and sounds to accompany the hiding and showing of windows.

The 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 (on Mac OS 8/9), and the necessary update and activate events to be generated.

Creating and Storing Windows

Mac OS 8.5 introduced 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 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.

Your application can store a window into a collection using the function StoreWindowIntoCollection. (This applies to any window, not just windows created using the new functions introduced with Mac OS 8.5.) Data associated with the window (for example, text) may also be stored into the same collection.

Using the 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 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 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 window.
kMovableAlertWindowClass	2L	Movable alert window.
kModalWindowClass	3L	Modal dialog window.
kMovableModalWindowClass	4L	Movable modal dialog window.
kFloatingWindowClass	5L	Floating window.
kDocumentWindowClass	6L	Document window or modeless dialog window. **Note:** The Window Manager assigns this class to windows created using the older window creation functions.
kUtilityWindowClass	8L	System-wide floating window.
kHelpWindowClass	10L	Help window.
kSheetWindowClass	11L	Sheet window. (Mac OS X only.)
kToolbarWindowClass	12L	Toolbar windows (above documents, below floating windows).
kPlainWindowClass	13L	Plain window (in document layer).
kOverlayWindowClass	14L	Transparent window which allows "screen" drawing via CoreGraphics. (Mac OS X only.)
kSheetAlertWindowClass	15L	Sheet windows for alerts. (Mac OS X only.)

kAltPlainWindowClass	16L	Alternate plain window (in document layer).
kAllWindowClasses	0xFFFFFFFF	For use with GetFrontWindowOfClass, FindWindowOfClass, GetNextWindowOfClass (see below).

Window Attribute Constants

Constant	Bit	Description
kWindowNoAttributes	0L	No attributes.
kWindowCloseBoxAttribute	1L << 0	Has close box/button.
kWindowHorizontalZoomAttribute	1L << 1	Has horizontal zoom box.
kWindowVerticalZoomAttribute	1L << 2	Has vertical zoom box.
kWindowFullZoomAttribute	kWindowVerticalZoomAttribute | kWindowHorizontalZoomAttribute	Has full zoom box/zoom button.
kWindowCollapseBoxAttribute	1L << 3	Has a collapse box/minimise button.
kWindowResizableAttribute	1L << 4	Has size box/resize control.
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.
kWindowNoShadowAttribute	1L << 21	No shadow. (Mac OS X only.)
kWindowHideOnSuspendAttribute	1L << 24	Window is automatically hidden and shown on, respectively, suspend and shown on resume. (Carbon only.)
kWindowStandardHandlerAttribute	1L << 25	Window should have standard window event handler installed. (Carbon only.)
kWindowHideOnFullScreenAttribute	1L << 26	Window is automatically hidden during fullscreen mode. (Carbon only.)
kWindowInWindowMenuAttribute	1L << 27	Window is automatically tracked in Window menu. (Document windows are automatically given this attribute.)
kWindowLiveResizeAttribute	1L << 28	Window supports live resizing. (Mac OS X only.)
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

The following functions are provided for accessing window information:

Function	Description
GetWindowClass	Obtains the class of a window.
GetWindowAttributes	Obtains the attributes of a window.
ChangeWindowAttributes	Change the attributes of a window.

IsValidWindowPtr	Reports whether a reference is a valid window reference.
FrontNonFloatingWindow	Returns a reference to the frontmost visible window that is not a floating window.
FindWindowOfClass	A version of the FindWindow function which limits the search to windows of one particular class. If a window is found at the specified point, but is not of the specified class, errWindowNotFound is returned and the value of outWindow is set to NULL.
GetFrontWindowOfClass	A more explicit version of the FrontWindow and FrontNonFloatingWindow functions.
GetNextWindowOfClass	A more explicit version of the function GetNextWindow.

Moving and Positioning Windows

Moving Windows

When your application wishes to move a window for a reason other than a user-instigated drag, it should use the function MoveWindowStructure or the earlier function MoveWindow. MoveWindow repositions a window's content region, whereas MoveWindowStructure repositions a window's structure region.

The 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 function GetWindowBounds obtains the size and position of the bounding rectangle of the specified window region.

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 systems with multiple monitors.

The 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	The window is centered on the screen that contains the menu bar.
kWindowCenterOnParentWindow	0x00000002	The window is centered on the parent window. If the window to is wider than the parent window, its left edge is aligned with the parent window's left edge.
kWindowCenterOnParentWindowScreen	0x00000003	The window is centered on the screen containing the parent window.
kWindowCascadeOnMainScreen	0x00000004	The window is placed just below the menu bar at the left edge of the main screen. Subsequent windows are placed relative to the first window in such a way that the frame of the preceding window remains visible behind the current window.
kWindowCascadeOnParentWindow	0x00000005	The window is placed below and to the right of the upper-left corner of the parent window in such a way that the frame of the parent window remains visible behind the current window.
kWIndowCascadeOnParentWindowScreen	0x00000006	The window is placed just below the menu bar at the left edge of the screen that contains the parent window. Subsequent windows are placed on the screen relative to the first window in such a way that the frame of the preceding window remains

		visible behind the current window.
kWindowAlertPositionOnMainScreen	0x00000007	The window is centered horizontally, and positioned vertically, on the screen that contains the menu bar in such a way that about one-fifth of the screen is above it.
kWindowAlertPositionOnParentWindow	0x00000008	The window is centered horizontally, and positioned vertically, in such a way that about one-fifth of the parent window is above it.
kWindowAlertPositionOnParentWindowScreen	0x00000009	The window is centered horizontally, and positioned vertically, in such a way that about one-fifth of the screen containing the parent window is above it.

These constants should not be confused with the older positioning specification constants (see Chapter 4), 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 function SetWRefCon has always allowed your application to associate a pointer to data with a reference to a window object. An alternative method of associating data with windows is to use the standard mechanism introduced with Mac OS 8.5. (Both methods, incidentally, are Carbon-compliant.)

The following functions are provided for associating data with windows:

Function	Description
SetWindowProperty	Associates data with a window.
GetWindowProperty	Gets data associated with a window.
GetWindowPropertySize	Gets the size of data associated with a window.
RemoveWindowProperty	Removes data associated with a window.

Adding To and Removing From the Update Region

The Mac OS 8.5 Window Manager provided enhanced functions for manipulating the update region. Unlike their pre-Mac OS 8.5 counterparts (InvalRect, InvalRgn, ValidRect, and ValidRgn, which are not included in the Carbon API), 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 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 following functions set 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	Gets 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	Gets the pattern to which a window's content region is redrawn.

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

Window Scrolling

The following functions scroll pixels within a window:

Function	Description
ScrollWindowRect ScrollWindowRegion	Scrolls pixels that are inside the specified rectangle (ScrollWindowRect) or region (ScrollWindowRegion). The pixels are shifted a distance of inHPixels horizontally and inVPixels vertically. The positive directions are to the right and down. The pixels that are shifted out of the specified window are not displayed, and the bits they represent are not saved. The exposed empty area created by the scrolling may be added to the update region or erased to the background colour/pattern of the window's graphics port.

The following constants may be passed in the inOptions parameter of these functions:

Constant	Meaning
kScrollWindowNoOptions	No options.
kScrollWindowInvalidate	Add the exposed area to the window’s update region.
kScrollWindowEraseToPortBackground	Erase the exposed area using the background colour/pattern of the window’s graphics port.

The Window Menu

Carbon introduced the system-managed Window menu, which is created using the function CreateStandardWindowMenu. After creating the menu you should add it to the menu list using InsertMenu. Menu items containing the titles of your application's windows will be automatically added to, and deleted from, the menu when those windows are created and closed. Floating windows will not be added to the menu.

It is not necessary to set the kWindowInWindowMenuAttribute attribute on your document windows in order for them to be added to the menu. Document windows are automatically given this attribute.

Using SetWindowAlternateTitle you can override the title displayed in the Window menu. You would ordinarily do this if the window title was not expressive enough.

Customising the Window Menu

You can insert your own items to the Window menu by searching for the item with command ID 'wldv' and inserting your items before that item (that is, immediately before the individual window items). 'wldv' is the command ID of the divider that separates the window commands from the individual window items.

A problem here is that, at the time of writing, the divider had the 'wldv' command ID on Mac OS X but not in CarbonLib.

You can append your own items at the end of the Window menu by searching for the item with command ID 'wlst' and appending your items after that item. 'wlst' is the command ID of a hidden menu item that marks the end of the individual window items.

Live Window Resizing

On Mac OS X, windows on which the kWindowLiveResizeAttribute attribute is set support live resizing. When this attribute is set, the window is continually redrawn while it is being resized, as opposed to just outlines of the window, title bar, and resize control being drawn. Full implementation of live resizing requires that

the contents of the content region also be continually redrawn as the window is being resized. This requires the use of the Carbon event model.

The demonstration program CarbonEvents2 (Chapter 17) demonstrates live resizing.

Main Constants, Data Types, and Functions

Constants

Window Class

```
KAlertWindowClass                = 1L
kMovableAlertWindowClass           = 2L
kModalWindowClass                 = 3L
kMovableModalWindowClass             = 4L
kFloatingWindowClass              = 5L
kDocumentWindowClass                 = 6L
kUtilityWindowClass              = 8L
kHelpWindowClass                  = 10L
kSheetWindowClass                 = 11L
kToolbarWindowClass               = 12L
kPlainWindowClass                = 13L
kOverlayWindowClass                = 14L
kSheetAlertWindowClass               = 15L
kAltPlainWindowClass              = 16L
kAllWindowClasses                = 0xFFFFFFFF
```

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
kWindowNoShadowAttribute              = 1L << 21
kWindowHideOnSuspendAttribute            = 1L << 24
kWindowStandardHandlerAttribute          = 1L << 25
kWindowHideOnFullScreenAttribute         = 1L << 26
kWindowInWindowMenuAttribute             = 1L << 27
kWindowLiveResizeAttribute             = 1L << 28
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
```

Window Scrolling

```
KScrollWindowNoOptions                 = 0
KScrollWindowInvalidate                = (1L << 0)
KscrollWindowEraseToPortBackground       = (1L << 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;
```

Window Scrolling

```
typedef UInt32 ScrollWindowOptions;
```

Functions

Floating Windows

```
Boolean   AreFloatingWindowsVisible(void);
```

Window Proxy Icons

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

Window Path Pop-Up Menus

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

Transitional Window Animations and Sounds

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

Creating and Storing Windows

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

Accessing Window Information

```
OSStatus  GetWindowClass(WindowRef window,WindowClass *outClass);
OSStatus  GetWindowAttributes(WindowRef window,WindowAttributes *outAttributes);
OSStatus  ChangeWindowAttributes(WindowRef window,WindowAttributes setTheseAttributes,
        WindowAttributes clearTheseAttributes);
Boolean   IsValidWindowRef(GrafPtr grafPort);
WindowRef FrontNonFloatingWindow(void);
OSStatus  FindWindowOfClass(const Point *where,WindowClass inWindowClass,
        WindowRef *outWindow,WindowPartCode *outWindowPart)
WindowRef GetFrontWindowOfClass(WindowClass inWindowClass,Boolean mustBeVisible);
WindowRef GetNextWindowOfClass(WindowRef inWindow,WindowClass inWindowClass,
        Boolean mustBeVisible);
```

Moving and Positioning Windows

```
OSStatus  MoveWindowStructure(WindowRef window,short hGlobal,short vGlobal);
OSStatus  SetWindowBounds(WindowRef window,WindowRegionCode regionCode,
        const Rect *globalBounds);
OSStatus  GetWindowBounds(WindowRef window,WindowRegionCode regionCode,Rect *globalBounds);
OSStatus  RepositionWindow(WindowRef window,WindowRef parentWindow,
        WindowPositionMethod method);
```

Associating Data With Windows

```
OSStatus  SetWindowProperty(WindowRef window,PropertyCreator propertyCreator,
        PropertyTag propertyTag,UInt32 propertySize,void *propertyBuffer);
```

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

Adding To and Removing From the Update Region

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

Setting Content Region Colour and Pattern

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

Window Scrolling

```
OSStatus  ScrollWindowRect(WindowRef inWindow,const Rect *inScrollRect,SInt16 inHPixels,
        SInt16 inVPixels,ScrollWindowOptions inOptions,RgnHandle outExposedRgn);
OSStatus  ScrollWindowRegion(WindowRef inWindow,RgnHandle inScrollRgn,SInt16 inHPixels,
        SInt16 inVPixels,ScrollWindowOptions inOptions,RgnHandle outExposedRgn);
```

Creating a Window Menu

```
OSStatus  CreateStandardWindowMenu(OptionBits inOptions,MenuRef *outMenu);
OSStatus  SetWindowAlternateTitle(WindowRef inWindow,CFStringRef inTitle);
```

Demonstration Program Windows2 Listing

```
// ****************************************************************************************************
// Windows2.c                                                    CLASSIC EVENT MODEL
// ****************************************************************************************************
//
// This program demonstrates the following Window Manager features and functions introduced
// with Mac OS 8.5:
//
// •  Creating floating windows and document windows using CreateNewWindow.
//
// •  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.
//
// •  Showing and hiding windows using TransitionWindow.
//
// •  Displaying window proxy icons.
//
// The program also demonstrates the creation of the system-managed Window menu introduced
// with Carbon and, on Mac OS X, a partial implementation of live window resizing.
//
// Those aspects of the newer Window Manager features not demonstrated in this program (full
// implementation of window proxy icons and window path pop-up menus) are demonstrated at the
// demonstration program Files (Chapter 18).
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  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 movable modal alert invoked when the user chooses the About
//    Windows2... item from the  Apple/Application menu.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// In addition, the program itself creates a 'wind' resource, and saves it to the resource
// fork of the file titled "Document", when the user chooses CreateNewWindow from the
// Document Windows menu.
//
// ****************************************************************************************************

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

#include <Carbon.h>

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

#define rMenubar       128
#define mFile          129
#define  iQuit         12
#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   gRunningOnX = false;
SInt16    gDocResFileRefNum;
WindowRef gColoursFloatingWindowRef;
WindowRef gToolsFloatingWindowRef;
Boolean   gDone;

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

void    main                    (void);
void    doPreliminaries         (void);
OSErr   quitAppEventHandler     (AppleEvent *,AppleEvent *,SInt32);
void    doEvents                (EventRecord *);
void    doMouseDown             (EventRecord *);
void    doUpdate                (EventRecord *);
void    doUpdateDocumentWindow  (WindowRef);
void    doActivate              (EventRecord *);
void    doActivateDocumentWindow (WindowRef,Boolean);
void    doOSEvent               (EventRecord *);
void    doAdjustMenus           (void);
void    doMenuChoice            (SInt32);
OSErr   doCreateNewWindow       (void);
OSErr   doSaveWindow            (WindowRef);
OSErr   doCreateWindowFromResource (void);
OSErr   doCreateFloatingWindows (void);
void    doCloseWindow           (WindowRef);
void    doErrorAlert            (SInt16);
void    doConcatPStrings        (Str255,Str255);
// ********************************************************************************** main

void  main(void)
{
  MenuBarHandle     menubarHdl;
  SInt32            response;
  MenuRef           menuRef;
  OSErr             osError;
  SInt16            numberOfItems, a;
  MenuCommand       menuCommandID;
  FSSpec            fileSpecTemp;
  EventRecord       eventStructure;
  SInt32            sleepTime;
  WindowRef         windowRef;
  docStructureHandle docStrucHdl;
  UInt32            actualSize;

  //
.....................................................................................................................................
..................... do preliminaries

  doPreliminaries();

  // ..................................................... set up menu bar and menus, customise Window menu if running on OS X

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

  CreateStandardWindowMenu(0,&menuRef);
  InsertMenu(menuRef,0);

  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
```

```
  if(menuRef != NULL)
  {
    DeleteMenuItem(menuRef,iQuit);
    DeleteMenuItem(menuRef,iQuit - 1);
    DisableMenuItem(menuRef,0);
  }

  gRunningOnX = true;
}


// ..... open resource fork of file "Windows2 Document" and store file reference number

osError = FSMakeFSSpec(0,0,"\pWindows2 Document",&fileSpecTemp);
if(osError == noErr)
  gDocResFileRefNum = FSpOpenResFile(&fileSpecTemp,fsWrPerm);
else
  doErrorAlert(osError);

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

if(osError = doCreateFloatingWindows())
  doErrorAlert(osError);

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

gDone = false;
sleepTime = GetCaretTime();

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

// ********************************************************************************* doPreliminaries

void  doPreliminaries(void)
{
 OSErr osError;

 MoreMasterPointers(128);
 InitCursor();
 FlushEvents(everyEvent,0);

 osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                      NewAEEventHandlerUPP((AEEventHandlerProcPtr) quitAppEventHandler),
                      0L,false);
 if(osError != noErr)
   ExitToShell();
}

// ********************************************************************************* doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
 OSErr    osError;
 DescType returnedType;
 Size     actualSize;

 osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,NULL,0,
                        &actualSize);
```

```
  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ********************************************************************************** doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  switch(eventStrucPtr->what)
  {
    case mouseDown:
      doMouseDown(eventStrucPtr);
      break;

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

    case updateEvt:
      doUpdate(eventStrucPtr);
      break;

    case activateEvt:
      doActivate(eventStrucPtr);
      break;

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

// ********************************************************************************** doMouseDown

void  doMouseDown(EventRecord *eventStrucPtr)
{
  WindowRef      windowRef;
  WindowPartCode partCode, zoomPart;
  SInt32         menuChoice;
  WindowClass    windowClass;
  BitMap         screenBits;
  Rect           portRect, mainScreenRect;
  Point          standardStateHeightAndWidth;

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

  switch(partCode)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

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

    case inContent:
      GetWindowClass(windowRef,&windowClass);
      if(windowClass == kFloatingWindowClass)
      {
        if(windowRef != FrontWindow())
          SelectWindow(windowRef);
        else
        {
          if(windowRef == gColoursFloatingWindowRef)
            ; // Appropriate action for Colours floating window here.
```

```
      else if(windowRef == gToolsFloatingWindowRef)
        ; // Appropriate action for Tools floating window here.
    }
  }
  else
  {
    if(windowRef != FrontNonFloatingWindow())
      SelectWindow(windowRef);
    else
      ; // Appropriate action for active document window here.
  }
  break;

case inDrag:
  DragWindow(windowRef,eventStrucPtr->where,NULL);
  break;

case inGoAway:
  GetWindowClass(windowRef,&windowClass);
  if(windowClass == kFloatingWindowClass)
  {
    if(TrackGoAway(windowRef,eventStrucPtr->where) == true)
      TransitionWindow(windowRef,kWindowZoomTransitionEffect,
                       kWindowHideTransitionAction,NULL);
  }
  else
    if(TrackGoAway(windowRef,eventStrucPtr->where) == true)
      doCloseWindow(windowRef);
  break;

case inGrow:
  ResizeWindow(windowRef,eventStrucPtr->where,NULL,NULL);
  GetWindowPortBounds(windowRef,&portRect);
  InvalWindowRect(windowRef,&portRect);
  break;

case inZoomIn:
case inZoomOut:
  mainScreenRect = GetQDGlobalsScreenBits(&screenBits)->bounds;
  standardStateHeightAndWidth.v = mainScreenRect.bottom - 100;
  standardStateHeightAndWidth.h = 600;

  if(IsWindowInStandardState(windowRef,&standardStateHeightAndWidth,NULL))
    zoomPart = inZoomIn;
  else
    zoomPart = inZoomOut;

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

// ********************************************************************************** doUpdate

void  doUpdate(EventRecord *eventStrucPtr)
{
  GrafPtr   oldPort;
  WindowRef windowRef;

  GetPort(&oldPort);
  windowRef = (WindowRef) eventStrucPtr->message;

  BeginUpdate(windowRef);

  SetPortWindowPort(windowRef);
  doUpdateDocumentWindow(windowRef);

  EndUpdate(windowRef);

  SetPort(oldPort);
}

// ********************************************************************** doUpdateDocumentWindow

void  doUpdateDocumentWindow(WindowRef windowRef)
{
  RgnHandle         visibleRegionHdl = NewRgn();
  Rect              contentRect;
```

```
  OSStatus           osError;
  UInt32             actualSize;
  docStructureHandle docStrucHdl;
  TEHandle           editStrucHdl;

   GetPortVisibleRegion(GetWindowPort(windowRef),visibleRegionHdl);
  EraseRgn(visibleRegionHdl);

  DrawGrowIcon(windowRef);

  if(!(osError = GetWindowProperty(windowRef,0,'docs',sizeof(docStrucHdl),&actualSize,
                              &docStrucHdl)))
  {
    GetWindowPortBounds(windowRef,&contentRect);
    InsetRect(&contentRect,3,3);
    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)
{
  WindowRef windowRef;
  Boolean   becomingActive;

  windowRef = (WindowRef) eventStrucPtr->message;
  becomingActive = ((eventStrucPtr->modifiers & activeFlag) == activeFlag);

  doActivateDocumentWindow(windowRef,becomingActive);
}

// ****************************************************************** doActivateDocumentWindow

void  doActivateDocumentWindow(WindowRef windowRef,Boolean becomingActive)
{
  docStructureHandle docStrucHdl;
  UInt32             actualSize;
  OSStatus           osError;

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

// ********************************************************************************** doOSEvent

void  doOSEvent(EventRecord *eventStrucPtr)
{
  switch((eventStrucPtr->message >> 24) & 0x000000FF)
  {
    case suspendResumeMessage:
      if((eventStrucPtr->message & resumeFlag) == 1)
        SetThemeCursor(kThemeArrowCursor);
      break;
  }
}

// ********************************************************************************** doAdjustMenus

void  doAdjustMenus(void)
{
  MenuRef       floatMenuRef;
  Boolean       isVisible;
  MenuItemIndex menuItem;

  isVisible = IsWindowVisible(gColoursFloatingWindowRef);
  GetIndMenuItemWithCommandID(NULL,'fcol',1,&floatMenuRef,&menuItem);
  CheckMenuItem(floatMenuRef,menuItem,isVisible);
```

```
  isVisible = IsWindowVisible(gToolsFloatingWindowRef);
  GetIndMenuItemWithCommandID(NULL,'ftoo',1,&floatMenuRef,&menuItem);
  CheckMenuItem(floatMenuRef,menuItem,isVisible);

  DrawMenuBar();
}

// ********************************************************************************** doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;
  OSErr         osError;
  MenuCommand   commandID;

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

  if(menuID == 0)
    return;

  osError = GetMenuItemCommandID(GetMenuRef(menuID),menuItem,&commandID);
  if(osError == noErr && commandID != 0)
  {
    switch(commandID)
    {
      case 'abou':
        Alert(rAboutAlert,NULL);
        break;

      case 'quit':
        gDone = true;
        break;

      case 'cwin':
        if(osError = doCreateNewWindow())
          doErrorAlert(osError);
        break;

      case 'cwir':
        if(osError = doCreateWindowFromResource())
          doErrorAlert(osError);
        break;

      case 'fcol':
        if(IsWindowVisible(gColoursFloatingWindowRef))
          TransitionWindow(gColoursFloatingWindowRef,kWindowZoomTransitionEffect,
                           kWindowHideTransitionAction,NULL);
        else
          TransitionWindow(gColoursFloatingWindowRef,kWindowZoomTransitionEffect,
                           kWindowShowTransitionAction,NULL);
        break;

      case 'ftoo':
        if(IsWindowVisible(gToolsFloatingWindowRef))
          TransitionWindow(gToolsFloatingWindowRef,kWindowZoomTransitionEffect,
                           kWindowHideTransitionAction,NULL);
        else
          TransitionWindow(gToolsFloatingWindowRef,kWindowZoomTransitionEffect,
                           kWindowShowTransitionAction,NULL);
        break;
    }
  }

  HiliteMenu(0);
}

// ********************************************************************** doCreateFloatingWindows

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

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

```
  if(!(osError = CreateNewWindow(kFloatingWindowClass,
                                 kWindowStandardFloatingAttributes |
                                 kWindowSideTitlebarAttribute,
                                 &contentRect,&gColoursFloatingWindowRef)))
  {
    if(pictureHdl = GetPicture(rColoursPicture))
      SetWindowPic(gColoursFloatingWindowRef,pictureHdl);

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

  if(osError != noErr)
    return osError;

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

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

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

  return osError;
}

// ************************************************************************** doCreateNewWindow

OSErr  doCreateNewWindow(void)
{
  Rect              contentRect;
  OSStatus          osError;
  WindowRef         windowRef;
  docStructureHandle docStrucHdl;
  Handle            textHdl;
  MenuRef           menuRef;

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

  do
  {
    if(osError = CreateNewWindow(kDocumentWindowClass,kWindowStandardDocumentAttributes,
                                 &contentRect,&windowRef))
      break;

    if(gRunningOnX)
      ChangeWindowAttributes(windowRef,kWindowLiveResizeAttribute,0);

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

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

    SetPortWindowPort(windowRef);
    UseThemeFont(kThemeSmallSystemFont,smSystemScript);

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

    OffsetRect(&contentRect,-contentRect.left,-contentRect.top);
    InsetRect(&contentRect,3,3);
    contentRect.right -= 15;
    contentRect.bottom -= 15;

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

```
    SetWTitle(windowRef,"\pCreateNewWindow");

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

    if(osError = doSaveWindow(windowRef))
      break;

  } while(false);

  if(osError)
  {
    if(windowRef)
      DisposeWindow(windowRef);

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

  return osError;
}

// ********************************************************************************** doSaveWindow

OSErr  doSaveWindow(WindowRef windowRef)
{
  SInt16             oldResFileRefNum;
  Collection         collection = NULL;
  OSStatus           osError;
  docStructureHandle docStrucHdl;
  UInt32             actualSize;
  Handle             flatCollectHdl, flatCollectResHdl, existingResHdl;

oldResFileRefNum = CurResFile();
UseResFile(gDocResFileRefNum);

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

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

  if(osError = GetWindowProperty(windowRef,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(gDocResFileRefNum);
    osError = ResError();
    if(osError != noErr)
      break;
  } while(false);

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

  UseResFile(oldResFileRefNum);

  return osError;
}

// ************************************************************** doCreateWindowFromResource

OSErr  doCreateWindowFromResource(void)
{
  SInt16             oldResFileRefNum;
  OSStatus           osError;
  WindowRef          windowRef;
  Collection         unflattenedCollection = NULL;
  Handle             windResHdl;
  docStructureHandle docStrucHdl;
  SInt32             dataSize = 0;
  Handle             textHdl;
  Rect               contentRect;

  oldResFileRefNum = CurResFile();
  UseResFile(gDocResFileRefNum);

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

    if(gRunningOnX)
      ChangeWindowAttributes(windowRef,kWindowLiveResizeAttribute,0);

    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;

  GetWindowPortBounds(windowRef,&contentRect);
  contentRect.right -= 15;
  contentRect.bottom -= 15;
  SetPortWindowPort(windowRef);
  UseThemeFont(kThemeSmallSystemFont,smSystemScript);
  (*docStrucHdl)->editStrucHdl = TENew(&contentRect,&contentRect);
  TEInsert(*textHdl,dataSize,(*docStrucHdl)->editStrucHdl);

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

  SetWTitle(windowRef,"\pCreateWindowFromResource");

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

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

  UseResFile(oldResFileRefNum);

  return osError;
}

// ********************************************************************************** doCloseWindow

void  doCloseWindow(WindowRef windowRef)
{
  OSStatus          osError;
  docStructureHandle docStrucHdl;
  UInt32            actualSize;

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

    if(osError = GetWindowProperty(windowRef,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(windowRef);
}
```

```
// ************************************************************************** doErrorAlert

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

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

  StandardAlert(kAlertStopAlert,theString,NULL,NULL,&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 Windows2 Comments

Two Window Manager features introduced with Mac OS 8.5 (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 18.

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.

- 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.

- Note the transitional animations when the document windows are opened and closed.

- Exercise the system-managed Window menu, noting the customisation of this menu when the program is run on Mac OS X.

defines

rWind represents the ID of the 'wind' resource created by the program.

typedefs

A document structure of type docStructure will be associated with 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

gDocResFileRefNum will be assigned the file reference number for the resource fork of the document file "Windows2 Document" included in the demo program's folder. gColoursFloatingWindowRef and gToolsFloatingWindowRef will be assigned references to the window objects for the floating windows.

main

The call to CreateStandardWindowMenu creates the system-managed Window menu, which is added to the menu list by the call to InsertMenu. If the program is running on Mac OS X, the next block customises the Window menu by searching for the item with the command ID 'wldv' (that is, the divider between the commands and the individual window items), inserting a divider and two custom items before that item, and assigning command IDs to those items. (At the time of writing, the divider did not have the 'wldv' command ID in CarbonLib.)

The resource fork of the file titled "Windows2 Document" is opened and the file reference number is saved to a global variable. The program will be saving a 'wind' resource to this file's resource fork.

CurResFile is called to set the application's resource fork as the current resource file.

The function doCreateFloatingWindows is called to create and show the floating windows.

In the next block (the main event loop), WaitNextEvent's sleep parameter is assigned the value returned by GetCaretTime. (GetCaretTime 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. This ensures that TEIdle, which causes the caret to blink, will be called at the correct interval.

When WaitNextEvent returns 0 with a null event, FrontNonFloatingWindow is called to obtain a reference to the front document window. If such a window exists, 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 insertion point caret to blink.

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. 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, TransitionWindow is called to hide the window. If the window is a non-floating window, the function doCloseWindow is called to close the window.

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 drawing of the content region of the non-floating windows.

GetWindowProperty is then called to retrieve the handle to the window's document structure, which, as previously stated, contains a handle to a TextEdit 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 calls in this section are incidental to the demonstration. TextEdit is addressed at Chapter 21.)

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.

GetWindowProperty is called to retrieve the handle to the window's document structure, which contains a handle to the TextEdit 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.

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.

If the user chose the first item in the Document Windows menu, the function doCreateNewWindow is called. If the user chose the second item, the function doCreateWindowFromResource is called. If either of these functions return an error, an error-handling function is called.

When an item in the Floating Windows menu is chosen, IsWindowVisible is called to determine the visibility state of the relevant floating window. 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. 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, 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 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).

if the program is running on Mac OS X, ChangeWindowAttributes is called to set the kWindowLiveResizeAttribute. This results in a partial implementation of live resizing.

NewHandle is then called to create a relocatable block for the document structure to be associated with the window. 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 SetPortWindowPort sets the window's graphics port as the current port and the call to UseThemeFont sets the window's font to the small system font.

The next three blocks load a 'TEXT' resource, insert the text into a TextEdit 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 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. 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 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, TransitionWindow is called to make the window visible (with animation).

To facilitate the demonstration of creating a window from a 'wind' resource (see the function doCreateWindowFromResource), a 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.

Firstly, the current resource file's file reference number is saved and the resource fork of the document titled "Windows2 Document" is made the current resource file.

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

GetWindowProperty retrieves the handle to the window's document structure.

The handle to the window's text is stored in the hText field of the TextEdit structure. The handle to the TextEdit 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 document file'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.

Finally, the saved resource file is made the current resource file.

doCreateWindowFromResource

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

Firstly, the current resource file's file reference number is saved and the resource fork of the document titled "Windows2 Document" is made the current resource file.

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 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 TextEdit structure.

The call to SetWindowProperty associates the document structure with the window, thus associating the TextEdit 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. 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 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, TransitionWindow is called to make the window visible (with animation).

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

Finally, the saved resource file is made the current resource file.

doCloseWindow

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

TransitionWindow is called to hide the window (with animation). GetWindowProperty is then called to retrieve a handle to the window's document structure, allowing the memory occupied by the TextEdit 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

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. eventFilter supports doErrorAlert.