14

MORE ON CONTROLS

Includes Demonstration Program Controls3

Introduction

Chapter 7 — Introduction to Controls introduced the subject of controls and addressed the basic controls (push buttons, checkboxes, radio buttons, scroll bars, and pop-up menu buttons), primary group boxes (test title variant) and user panes. This chapter revisits group boxes and user panes and addresses the many remaining controls, all but one of which were introduced with Mac OS 8 and the Appearance Manager.

The controls addressed in this chapter are as follows:

Bevel button	Image well	Tab	Edit text	Slider
Group boxes	Clock	Progress indicator	Little arrows	Disclosure triangle
Picture	Icon	Window header	Placard	Static text
Separator line	Pop-up arrow	Radio Group	Asynchronous arrows	User pane
Scrolling Text Box	(introduced with Mac OS 8.5)			

The one remaining control (the list box) will be addressed at Chapter 20 — Lists and Custom List Definition Functions.

The progress indicator will be described in this chapter and the indeterminate variant demonstrated in the associated demonstration program. However, because the use of the determinate variant is ordinarily associated with the matter of scanning for a Command-period event, the demonstration of that particular variant has been held over to the demonstation program associated with Chapter 23 — Miscellany.

Preamble — Review of Control Basics

Recall from Chapter 7 that:

- Control definition functions (**CDEF**s) determine the appearance and behaviour of controls.

- Two new features introduced with Mac OS 8 and the Appearance Manager were **embedding** and **latency**. Embedding hierarchies have implications for drawing order and hit testing.

- Another feature introduced with Mac OS 8 and the Appearance Manager was read and write access to the various attributes of a control. Each piece of information that a particular CDEF allows access to is referenced by a control data tag. **Control data tag constants** are passed in the inTagName parameter of the getter and setter functions SetControlData and GetControlData.

- FindControl is used to determine whether a mouse-down event occurred in a control and, if so, which control. FindControl returns a **control part code** identifying the part of the control in which the mouse-down occurred. kControlNoPart (0) is returned if the mouse-down occurred where no enabled control exists. TrackControl or HandleControlClick are used to handle user interaction with a control as long as the user holds the mouse button down. These two functions return kControlNoPart if the cursor is outside the control or control part when the mouse button is released or the relevant control part code if the user releases the mouse button while the cursor is still inside the control or control part.

- The font for any control can be set independently of the system or window font.

- The pop-up menu button differs from the other basic controls in that the control's **initial**, **minimum**, and **maximum** values do not actually represent initial, minimum and maximum values as such. For example, the minimum value field of the 'CNTL' resource for a pop-up button is used to specify the resource ID of the 'MENU' resource utilised by the control.

The use of the initial, minimum, and maximum fields for purposes other than control values as such also applies to most of the controls addressed in this chapter.

More on Embedding

As stated at Chapter 8 — Dialogs and Alerts, if the kDialogFlagsUseControlHierarchy feature bit is set in a dialog's 'dlgx' resource, the Dialog Manager creates a root control in the dialog box and establishes an embedding hierarchy.

The Dialog Manager uses AutoEmbedControl to position dialog items in an embedding hierarchy based on both visual containment and the order of the items in the item list. As items are added to a dialog box during creation, controls that already exist in the window will be containers for new controls if they both visually contain the control and are themselves embedder controls. For this reason, you should place the largest embedder controls at the beginning of the item list. As an example, the Dialog Manager will automatically embed radio buttons in a group box if, firstly, the radio buttons visually "fit" inside the group box and, secondly, the group box precedes the radio buttons in the item list.

Control Descriptions

In the following, those control types and constants introduced with Mac OS 8.5 appear on a dark gray background.

Bevel Buttons

A bevel button is a rectangular control with a bevelled edge which can include text, an icon, a picture, or a combination of text and an icon or picture. Bevel buttons mimic the behaviour of other button types: they can behave like push buttons; in sets, they can behave like radio buttons or checkboxes; if a menu is attached, they behave like pop-up menu buttons. Typical bevel buttons are shown at Fig 1.

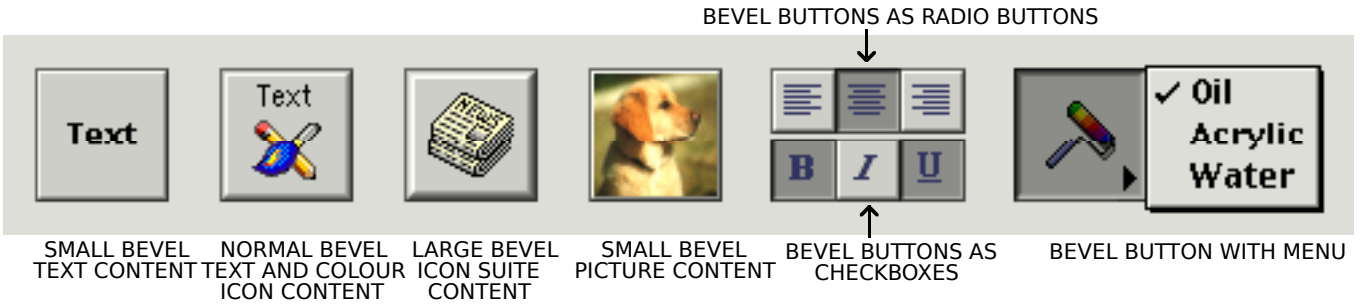

FIG 1 - TYPICALBEVEL BUTTONS

Variants and Control Definition IDs

The bevel button CDEF resource ID is 2. The six available variants and their control definition IDs are as follows:

Variant	Var Code	Control Definition ID	
With small (2 pixel wide) bevel. (Pop-up menu, if any, below.)	0	32	kControlBevelButtonSmallBevelProc
With medium (3 pixel wide) bevel. (Pop-up menu, if any, below.)	1	33	kControlBevelButtonNormalBevelProc
With large (4 pixel wide) bevel. (Pop-up menu, if any, below.)	2	34	kControlBevelButtonLargeBevelProc
With small (2 pixel wide) bevel. Pop-up menu on right.	4	36	KControlBevelButtonSmallBevelProc + kControlBevelButtonMenuOnRight
With medium (3 pixel wide) bevel. Pop-up menu if any on right.	5	37	KControlBevelButtonNormalBevelProc + kControlBevelButtonMenuOnRight
With large (4 pixel wide) bevel. Pop-up menu if any on right.	6	38	KControlBevelButtonLargeBevelProc + kControlBevelButtonMenuOnRight

For bevel buttons containing text, if the constant kControlUsesOwningWindowsFontVariant is added to the variation code, the text will appear in the window's font.

Control Values

The following lists the initial, minimum, and maximum value settings for bevel buttons:

Control Value	Content
Initial	If you wish to attach a menu, the menu ID. If no menu, 0.
Minimum	High byte specifies behaviour (see Bevel Button Behaviour Constants, and Bevel Button Menu Constants, below). Low byte specifies content type (see Bevel Button and Image Well Content Type Constants, below).
Maximum	Resource ID of bevel button’s content if resource-based.

Note that bevel buttons have two values: the value of the bevel button and, if a menu has been attached, the value of the menu.

Bevel Button Behaviour Constants

The following **bevel button behaviour constants** apply to the high byte of a bevel button's minimum value. For bevel buttons created programmatically using NewControl, these constants are passed in the high byte of the minimumValue parameter.

Constant	Value	Description

kControlBehaviorPushbutton	0	The bevel button pops up after being clicked. This constant is used when push button behaviour is required.
kControlBehaviorToggles	$0100	The bevel button toggles state automatically when clicked. This constant is used when checkbox or radio button behaviour is required.
kControlBehaviorSticky	$0200	Once clicked, the bevel button stays down until your application sets the control's value to 0. This behaviour is useful in tool palettes.
kControlBehaviorOffsetContents	$8000	Bevel button contents are offset (one pixel down and to the right) when button is pressed. (Some users consider that this behaviour gives a bevel button a more realistic "feel".)

Bevel Button Menu Constants

The following **bevel button menu constants** apply to the high byte of a bevel button's minimum value. For bevel buttons created programmatically using NewControl, these constants are passed in the high byte of the minimumValue parameter.

Constant	Value	Description
kControlBehaviorCommandMenu	$2000	If this bit is set, the menu contains commands, not choices, and should not be marked with a checkmark. If this bit is set, it overrides the kControlBehaviorMultiValueMenu bit.
kControlBehaviorMultiValueMenu	$4000	If this bit is set, the menus are multi-valued. The bevel button does not maintain the menu value as it normally would (requiring that only one item is selected at a time). This allows the user to toggle entries in a menu and have multiple items checked. In this mode, the menu value accessed using GetControlData with the kControlBevelButtonMenuValueTagwill return the value of the last menu item selected.

Bevel Button and Image Well Content Type Constants

The following **bevel button and image well content type constants** apply to the low byte of a bevel button's minimum value. For bevel buttons created programmatically using NewControl, these constants are passed in the low byte of the minimumValue parameter. You can also use these constants in the contentType field of the **bevel button and image well content structure** (see below). You can then pass a pointer to this structure in the inBuffer parameter of GetControlData and SetControlData to get and set the resource ID (for resource-based content) or handle (for handle-based content) of a colour icon, icon suite, or picture in a bevel button.[1]

Constant	Value	Description
kControlContentTextOnly	0	Content type is text only. This constant is passed in the contentType field of the bevel button and image well content structure if the content is text only. The variation code kControlUsesOwningWindowsFontVariant applies when text content is used.
kControlContentIconSuiteRes	1	Content type uses an icon suite resource ID. The resource ID of the icon suite resource you wish to display should be in the resID field of the bevel button and image well content structure.
kControlContentCIconRes	2	The resource ID of the colour icon resource you wish to display should be in the resID field of the bevel

[1] Note that resource-based content is owned by the control, while handle-based content is owned by you. The control definition function will not dispose of handle-based content. If you replace handle-based content with resource-based content on the fly, you must dispose of the handle properly to avoid a memory leak.

		button and image well content structure.
kControlContentPictRes	3	Content type is a picture resource ID. The resource ID of the picture resource you wish to display should be in the resID field of the bevel button and image well content structure.
kControlContentIconSuiteHandle	129	Content type is an icon suite handle. The handle of the icon suite you wish to display should be in the iconSuite field of the bevel button and image well content structure.
kControlContentCIconHandle	130	Content type uses a colour icon handle. The handle of the colour icon you wish to display should be in the cIconHandle field of the bevel button and image well content structure.
kControlContentPictHandle	131	Content type uses a picture handle. The handle of the picture you wish to display should be in the picture field of the bevel button and image well content structure.
kControlContentIconRef	132	Reserved.

Control Data Tag Constant — Content Type

The control data tag constant relevant to content type in bevel buttons is as follows:

Control Data Tag Constant	Meaning and Data Type Returned or Set
kControlBevelButtonContentTag	Gets or sets a bevel button's content type for drawing. (See Bevel Button and Image Well Content Type Constants, above. See also The Bevel Button and Image Well Structure, below.) **Data type returned or set:** ControlButtonContentInfo **structure**

Bevel Button Alignment and Placement Constants, and Associated Control Data Tag Constants

By calling SetControlData with certain control data tag constants, and with certain constants passed in the inData parameter, you can specify the alignment of icon suites, colour icons, pictures, and text in a bevel button, and you can specify the placement of text in relation to an icon suite, colour icon, or picture. By calling GetControlData with these constants you can also ascertain alignment and placement.

Bevel Button Graphic Alignment Constants

The following constants are used to specify the alignment of icon suites, colour icons, and pictures.

Constant	Value	Description
kControlBevelButtonAlignSysDirection	-1	Bevel button graphic is aligned according to the system default script direction (only left or right).
kControlBevelButtonAlignCenter	0	Graphic is aligned centre.
kControlBevelButtonAlignLeft	1	Graphic is aligned left.

kControlBevelButtonAlignRight	2	Graphic is aligned right.
kControlBevelButtonAlignTop	3	Graphic is aligned top.
kControlBevelButtonAlignBottom	4	Graphic is aligned bottom.
kControlBevelButtonAlignTopLeft	5	Graphic is aligned top left.
kControlBevelButtonAlignBottomLeft	6	Graphic is aligned bottom left.
kControlBevelButtonAlignTopRight	7	Graphic is aligned top right.
kControlBevelButtonAlignBottomRight	8	Graphic is aligned bottom right.

Bevel Button Text Alignment Constants

The following constants are used to specify the alignment of text.

Constant	Value	Description
kControlBevelButtonAlignTextSysDirection	0	Text is aligned according to the current script direction (left or right).
kControlBevelButtonAlignTextCenter	1	Text is aligned centre.
kControlBevelButtonAlignTextFlushRight	-1	Text is aligned flush right.
kControlBevelButtonAlignTextFlushLeft	-2	Text is aligned flush left.

Bevel Button Text Placement Constants

The following constants are used to specify the placement of text in relation to an icon suite, colour icon, or picture.

Constant	Value	Description
kControlBevelButtonPlaceSysDirection	-1	Text is placed according to the system default script direction.
kControlBevelButtonPlaceNormally	0	Text is centred.
kControlBevelButtonPlaceToRightOfGraphic	1	Text is placed to the right of the graphic.
kControlBevelButtonPlaceToLeftOfGraphic	2	Text is placed to the left of the graphic.
kControlBevelButtonPlaceBelowGraphic	3	Text is placed below the graphic.
kControlBevelButtonPlaceAboveGraphic	4	Text is placed above the graphic.

Control Data Tag Constants — Alignment and Placement

The control data tag constants relevant to the alignment and placement of graphics and text in bevel buttons are as follows:

Control Data Tag Constant	Meaning and Data Type Returned or Set
kControlBevelButtonTextAlignTag	Gets or sets the alignment of text in a bevel button. (See Bevel Button Text Alignment Constants, above.) **Data type returned or set:** ControlButtonTextAlignment
kControlBevelButtonGraphicAlignTag	Gets or sets the alignment of graphics in a bevel button in relation to any text the button may contain. (See Bevel Button Graphic Alignment Constants.) **Data type returned or set:** ControlButtonGraphicAlignment
kControlBevelButtonTextPlaceTag	Gets or sets the placement of a bevel button's text. (See Bevel Button Text Placement Constants, above.) **Data type returned or set:** ControlButtonTextPlacement
kControlBevelButtonTextOffsetTag	Gets or sets the number of pixels that text is offset in a bevel button from the button's left or right edge. This is used with left, right, or system justification, but it is ignored when the text is centre aligned. **Data type returned or set:** SInt16
kControlBevelButtonGraphicOffsetTag	Gets or sets the horizontal and vertical amounts that a graphic element contained in a bevel button is offset from the button's edges. This value is ignored

	when the graphic is specified to be centred on the button. **Data type returned or set:** Point

The Bevel Button and Image Well Content Structure

As previously stated, you can use bevel button and image well content type constants in the contentType field of the **bevel button and image well content structure** and you can then pass a pointer to this structure in the inBuffer parameter of GetControlData and SetControlData. The bevel button and image well content structure is as follows:

```
ControlButtonContentInfo = RECORD
    contentType:        ControlContentType;
        CASE INTEGER OF
        0: (
                    resID:              SInt16;
                    );
        1: (
                    cIconHandle:        CIconHandle;
                    );
        2: (
                    iconSuite:          Handle;
                    );
        3: (
                    iconRef:            Handle;
                    );
        4: (
                    picture:            PicHandle;
                    );
    END;

ControlButtonContentInfoPtr = ^ControlButtonContentInfo;
```

Field Descriptions

contentType — Specifies the bevel button or image well content type and whether the content is text-only, resource-based, or handle-based. (See Bevel Button and Image Well Content Type Constants, above.) The value specified in the contentType field determines which of the other fields in the structure are used.

resID — If the content type specified in the contentType field is kControlContentIconSuiteRes, kControlContentCIconRes, or kControlContentPictRes, this field contains the resource ID of a picture, colour icon, or icon suite resource.

cIconHandle — If the content type specified in the contentType field is kControlContentCIconHandle, this field contains a handle to a colour icon.

iconSuite — If the content type specified in the contentType field is kControlContentIconSuiteHandle, this field contains a handle to an icon suite.

iconRef — Reserved.

picture — If the content type specified in the contentType field is kControlContentPictHandle, this field contains a handle to a picture.

Other Control Data Tag Constants

The remaining control data tag constants relevant to bevel buttons are as follows:

Control Data Tag Constant	Meaning and Data Type Returned or Set
kControlBevelButtonCenterPopUpGlyphTag	Gets or sets the position of the pop-up arrow in a bevel button when a pop-up menu is attached. **Data type returned or set:** Boolean. If true, glyph is vertically centred on the right; if false, glyph is on the bottom right.
kControlBevelButtonTransformTag	Gets or sets a transform that is added to the

	standard transform of a bevel button. See the Icons section at Chapter 13 — Offscreen Graphics Worlds, Pictures, Cursors, and Icons.) **Data type returned or set:** IconTransformType
kControlBevelButtonMenuValueTag	Gets the menu value for a bevel button with an attached menu. (See "Bevel Button Menu Constants, above.) **Data type returned or set:** SInt16
kControlBevelButtonMenuHandleTag	Gets the menu handle for a bevel button with an attached menu. **Data type returned or set:** MenuHandle
kControlBevelButtonLastMenuTag	Gets the menu ID of the last menu selected in the submenu or main menu. **Data type returned or set:** Boolean
kControlBevelButtonMenuDelayTag	Gets or sets the delay (in number of ticks) before the menu is displayed. **Data type returned or set:** SInt32
kControlBevelButtonScaleIconTag	Gets or sets whether, when the proper icon size is unavailable, the icon should be scaled. For use only with icon suites or the IconRef data type. **Data type returned or set:** Boolean. If true, indicates that if an icon of the ideal size is not available a larger or smaller icon should be scaled. If false, no scaling should occur; instead, a smaller icon should be drawn or a larger icon clipped.

With regard to the kControlBevelButtonMenuDelayTag constant, setting a delay before the menu is displayed in a bevel button with sticky behaviour is useful for providing option sets in tool palettes. You can set up the bevel button so that:

- If the user clicks it once, it simply turns on the function represented by the button.
- If the user presses it for longer than the user-set double-click time, it displays a pop-up menu which offers further options for that function.

Control Part Codes

The following control part codes are relevant to bevel buttons:

Constant	Value	Description
kControlNoPart	0	For bevel buttons with a menu attached, this part code indicates that either the mouse was released outside the bevel button and menu or that the button was disabled.
kControlMenuPart	2	For bevel buttons with a menu attached, this part code indicates that the event occurred in a menu item of the bevel button.
kControlButtonPart	10	For bevel buttons with a menu attached, this part code indicates that the event occurred in the button but not in the attached menu.

Bevel Button States

Bevel buttons can exist in five active states and two disabled states. The active states are off, pressed (was off), on, pressed (was on), and mixed.[2] The mixed state is used, where appropriate, when the bevel button is behaving as a checkbox or radio button. Disabled bevel buttons can be shown as off or on.

[2] Under platinum appearance, both pressed states look the same, but this may change in future themes.

Image Wells

Image wells are used to display non-text visual content, such as icons or pictures. They have a two-pixel-wide rectangular frame and have a recessed appearance with a white background fill. Image wells are controlled in much the same way as bevel buttons, but with fewer options and states. They should not be used in place of push buttons or bevel buttons. Typical image wells are shown at Fig 2.

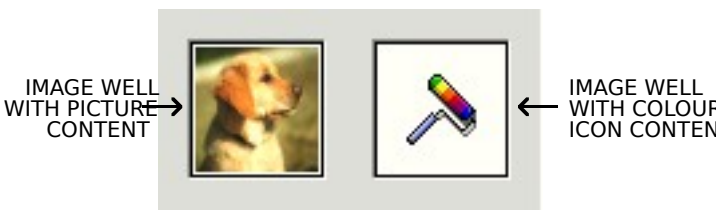

FIG 2 - TYPICAL IMAGE WELLS

Variants and Control Definition IDs

The image well CDEF resource ID is 11. The two available variants and their control definition IDs are as follows:

Variant	Var Code	Control Definition ID	
Image well.	0	176	kControlImageWellProc
Image well with autotracking.	1	177	kControlImageWellAutoTrackProc

Future versions will support drag-and-drop functionality.

Control Values

Control Value	Content
Initial	Resource ID of image well's content ('cicn', 'PICT' or icon suite).
Minimum	Content type. (See Bevel Button and Image Well Content Type Constants, above.) After the image well is created, reset to 0.
Maximum	Ignored. Reset to 2 after creation.

Bevel Button and Image Well Content Type Constants and The Bevel Button and Image Well Content Structure

The bevel button and image well content type constants (see Bevel Buttons, above) apply to an image well's minimum value. For image wells created programmatically using NewControl, these constants are passed in the minimumValue parameter.

You can also use these constants in the contentType field of the bevel button and image well content structure (see Bevel Buttons, above). You can then pass a pointer to this structure in the inBuffer parameter of GetControlData and SetControlData to get and set the resource ID (for resource-based content) or handle (for handle-based content) of a colour icon, icon suite, or picture in an image well.

Control Data Tag Constants

Control Data Tag Constant	Meaning and Data Type Returned or Set
kControlImageWellContentTag	Gets or sets the content for an image well. (See The Bevel Button and Image Well Content Structure, above.) **Data type returned or set:** ControlButtonContentInfo **structure**
kControlImageWellTransformTag	Gets or sets a transform that is added to the standard transform of an image well. (See the Icons section at Chapter 13 — Offscreen Graphics Worlds,

	Pictures, Cursors, and Icons.)
	Data type returned or set: IconTransformType

Control Part Codes

Constant	Value	Description
kControlImageWellPart	26	Event occurred in an image well.

Tab — Embedding Control

The tab control is an embedding control which provides a convenient way to present information in a multi-page format. The tab control is distinguished by the visual appearance of folder tabs. The user selects the desired "page" by clicking the appropriate tab, which highlights and displays its page.

The tab control supports one row of tabs running along the top (see Fig 3). You specify the names and icons that label the tabs.

FIG 3 - TAB CONTROL

The appearance of the content area of a tab control (also known as a pane) depends on where it is used. In a control panel, the sides of the pane should be tucked under the edge of the content region by one pixel. In a modal dialog box, the left and right sides of the pane should be inset two pixels from the edge of the dialog box's content region. This small distinction helps emphasise the fact that the tab is part of a dialog box. (See Fig 3.)

Controls such as push buttons, radio buttons, scroll bars, etc., may be used within a tab control, and should be embedded within an individual pane. Controls whose effect is intended to be global (that is, their setting are intended to affect all the panes in a set of tabs) should be located outside the tab control. It is important that you make such distinctions unambiguous to the user through clear, specific labelling and placement.

The tab information ('tab#') resource is used to provide the tab names and icon suite IDs.

Variants and Control Definition IDs

The tab control CDEF resource ID is 8. The two available variants and their control definition ID are follows:

Variant	Var Code	Control Definition ID
Normal tab control.	0	128 kControlTabLargeProc
Small tab control.	1	129 kControlTabSmallProc

Control Values

Control Value	Content
Initial	Resource ID of the 'tab#' resource you are using to hold tab information. Reset to the minimum setting after creation. A value of 0 indicates not to read a 'tab#' resource. (See The Tab Information Structure, below.)

Minimum	Ignored. Reset to 1 after creation.
Maximum	Ignored. Reset to the number of individual tabs in the tab control after creation.

Control Data Tag Constants

Control Data Tag Constant	Meaning and Data Type Returned or Set
kControlTabContentRectTag	Gets the content rectangle of a tab control. **Data type returned:** Rect
kControlTabEnabledFlagTag	Enables or disables a single tab in a tab control. **Data type returned or set:** Boolean; if true, enabled; if false, disabled.
kControlTabInfoTag	Gets or sets information for a tab in a tab control. (See The Tab Information Structure, below.) **Data type returned or set:** ControlTabInfoRec

The Tab Information Structure

If you are not creating a tab control with a 'tab#' resource, you can call SetControlMaximum to set the number of tabs in a tab control and then call SetControlData with the kControlTabInfoTag to set the information for an individual tab in a tab control. The tab information structure passed in the SetControlData call is as follows:

```
ControlTabInfoRec = RECORD
    version:        SInt16;     { Version of this structure. }
    iconSuiteID:    SInt16;     { Icon suite to use. Zero indicates no icon. }
    name:           Str255;     { Name to be displayed on the tab. }
    END;
```

Field Descriptions

version — A signed 16-bit integer indicating the version of the tab information structure. The only currently available version value is 0.

iconSuiteID — A signed 16-bit integer indicating the ID of an icon suite to be used for the tab label. Pass 0 for no icon.

name — A string specifying the title to be used for the tab label.

The Tab Information Resource

You can use a tab information resource to specify the icon suite ID and name of each tab in a tab control. A tab information resource is a resource of type 'tab#'. All tab information resources must have resource ID numbers greater than 127. The Control Manager uses the information you specify to provide additional information to the corresponding tab control. Figure 4 shows the structure of a compiled 'tab#' resource.

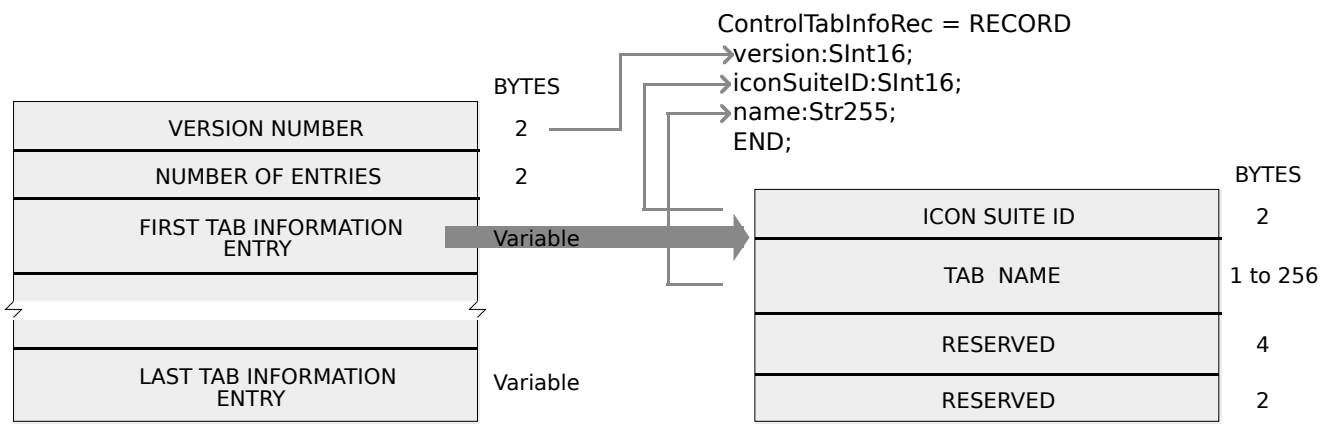

FIG 4 - STRUCTURE OF A COMPILED TAB INFORMATION ('tab#') RESOURCE

The following describes the fields of a compiled 'tab#' resource and one of its tab information entries:

Field	Description
VERSION NUMBER	An integer specifying the version of the resource.
NUMBER OF ENTRIES	An integer that specifies the number of tab information entries in the resource.
ICON SUITE ID	Icon suite resource ID.
TAB NAME	A variable-length string indicating the tab name.

Edit Text

Edit text controls are rectangular areas in which the user enters text or modifies existing text. Edit text controls supports keyboard focus and a password entry variant is available. Fig 5 shows a typical edit text control.

FIG 7 - EDIT TEXT FIELD

Edit text controls can have an application-defined key filter function attached to filter key strokes or modify them on return.

Variants and Control Definition IDs

The edit text field CDEF resource ID is 17. The three available variants and their control definition IDs are as follows:

Variant	Var Code	Control Definition ID	
Edit text control for windows. This control maintains its own TEHandle.	0	272	kControlEditTextProc
Edit text control for dialog boxes. This control uses the dialog box common TEHandle.	1	273	kControlEditTextDialogProc
Edit text control for passwords. This control is supported by the Script Manager. Password text can be accessed via the kEditTextPasswordTag constant. (See Control Data Tag Constants, below.)	2	274	kControlEditTextPasswordProc
Edit text control for inline input. This control supports 2-byte script systems.	4	276	kControlEditTextInlineInputProc

Control Values

Control Value	Content
Initial	Reserved. Set to 0.
Minimum	Reserved. Set to 0.
Maximum	Reserved. Set to 0.

Control Data Tag Constants

Control Data Tag Constant	Meaning and Data Type Returned or Set
kControlEditTextTextTag	Gets or sets text in an editable text control. **Data type returned or set:** character buffer
kControlEditTextTEHandleTag	Gets a handle to a text edit structure. **Data type returned or set:** TEHandle

kControlEditTextSelectionTag	Gets or sets the selection in an editable text control. **Data type returned or set:** ControlEditTextSelectionRec structure. (See The Edit Text Selection Structure, below.)
kControlEditTextPasswordTag	Gets clear password text from an editable text control, that is, the text of the actual password typed, not the bullet text. **Data type returned or set:** character buffer
kControlKeyFilterTag	Gets or sets a key filter function. **Data type returned or set:** ControlKeyFilterUPP
kControlEditTextStyleTag	Gets or sets the font style. **Data type returned or set:** ControlFontStyleRec
KControlEditTextLockedTag	Gets or sets whether the text is currently editable. **Data type returned or set:** Boolean. **If** true**, text is locked. If** false**, text is editable**
kControlEditTextValidationProcTag	Gets or sets a universal procedure pointer to a callback function which can be used to validate editable text after an operation that changes the text, such as a cut or paste. **Data type returned or set:** ControlEditTextValidationUPP

Control Part Codes

Constant	Value	Description
kControlEditTextPart	5	Event occurred in an Edit text control.

The Edit Text Selection Structure

You can pass a pointer to the edit text selection structure to GetControlData and SetControlData to access and set the current selection range in an edit text control. An edit text selection structure is of type ControlEditTextSelectionRec:

```
ControlEditTextSelectionRec = RECORD
    {  Structure for getting the edit text selection  }
    selStart:          SInt16;
    selEnd:            SInt16;
    END;

ControlEditTextSelectionPtr = ^ControlEditTextSelectionRec;
```

Field Descriptions

selStart — A signed 16-bit integer indicating the beginning of the edit text selection.

selEnd — A signed 16-bit integer indicating the end of the edit text selection.

Sliders

A slider control consists of a **slider bar**, which displays a range of allowable values, and an **indicator**, which marks the current setting. The user can drag the indicator to set a new value within the range.

Sliders can be horizontal or vertical. The indicator can point in any direction or be nondirectional. Typical sliders are shown at Fig 8.

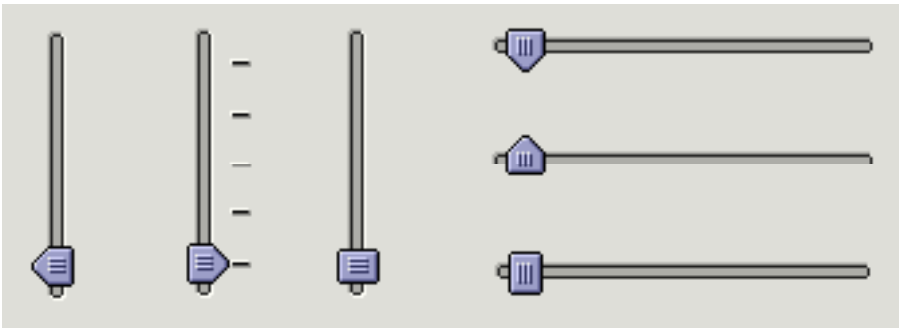

FIG 8 - SLIDERS

A slider can display tick marks to represent increments within the range of values. You can specify the number of tick marks required. If you use tick marks, they are drawn appropriately for the direction of the slider. You should ensure that tick marks are labelled so that they clearly indicate the effect of moving the indicator.

A live feedback variant is available. This variant continually updates the value of the control as the indicator is dragged, as opposed to the standard behaviour of updating the value only when the mouse button is released.

Variants and Control Definition IDs

The slider CDEF resource ID is 3. The ten available variants and their control definition IDs are as follows:

Variant	Var Code	Control Definition ID
Slider. If the slider is horizontal, the indicator points down, and if the slider is vertical, the indicator points right.	0	48 kControlSliderProc
Slider with live feedback. The value of the control is updated automatically by the Control Manager before your action function is called. If no application-defined action function is supplied, the slider draws ghosted image of the indicator as the user moves it.	1	49 kControlSliderProc + kControlSliderLiveFeedback
Slider with tick marks. The control rectangle must be large enough to include the tick marks.	2	50 kControlSliderProc + kControlSliderHasTickMarks
Slider with live feedback and tick marks.	3	51 kControlSliderProc + kControlSliderLiveFeedback + kControlSliderHasTickMarks
Slider with indicator reversed. . If the slider is horizontal, the indicator points up, and if the slider is vertical, the indicator points left.	4	52 kControlSliderProc + kControlSliderReverseDirection
Slider with live feedback and indicator reversed.	5	53 kControlSliderProc+ kControlSliderLiveFeedback + kControlSliderReverseDirection
Slider with tick marks and indicator reversed.	6	54 kControlSliderProc + kControlSliderHasTickMarks + kControlSliderReverseDirection
Slider with live feedback, tick marks and indicator reversed.	7	55 kControlSliderProc + kControlSliderLiveFeedback + kControlSliderHasTickMarks + kControlSliderReverseDirection
Slider with a rectangular, non-directional indicator.	8	56 kControlSliderProc + kControlSliderNonDirectional
Slider with live feedback and a rectangular, non-directional indicator.	9	57 kControlSliderProc + kControlSliderLiveFeedback + kControlSliderNonDirectional

Control Values

Control Value	Content
Initial	Appropriate value between –32768 and 32768. For the tick mark variant, the number of ticks required. Reset to the minimum setting after creation.
Minimum	–32768 to 32768.
Maximum	–32768 to 32768. When the maximum setting is equal to the minimum setting, the slider is inactive.

Control Part Codes

Constant	Value	Description
kControlIndicatorPart	129	Event occurred in the indicator.

Group Boxes — Embedding Control

Group boxes are embedding controls used to associate, isolate, and distinguish groups of related items in a dialog box. You can embed other controls, such as radio groups, checkboxes, and pop-up menu buttons, within group boxes.

Group boxes are defined as either primary or secondary. The main visual distinction between the two types is their border: primary group box border lines are two pixels wide with an etched look; secondary group box borders are one pixel wide.

You can use any of four titling options for the border of a group box. The group box can be untitled or it can have a text title, a pop-up menu title, or a checkbox title. Group boxes with a pop-up menu title are useful for displaying a variety of related settings in a limited space. Group boxes with a checkbox title are useful for indicating that a group of settings may be deactivated by the user.

Secondary group boxes are generally used for nesting and grouping together subsidiary information. Secondary group boxes should not be used in lieu of primary group boxes, since this produces an inconsistent appearance which could confuse users.

Typical group boxes are shown at Fig 9.

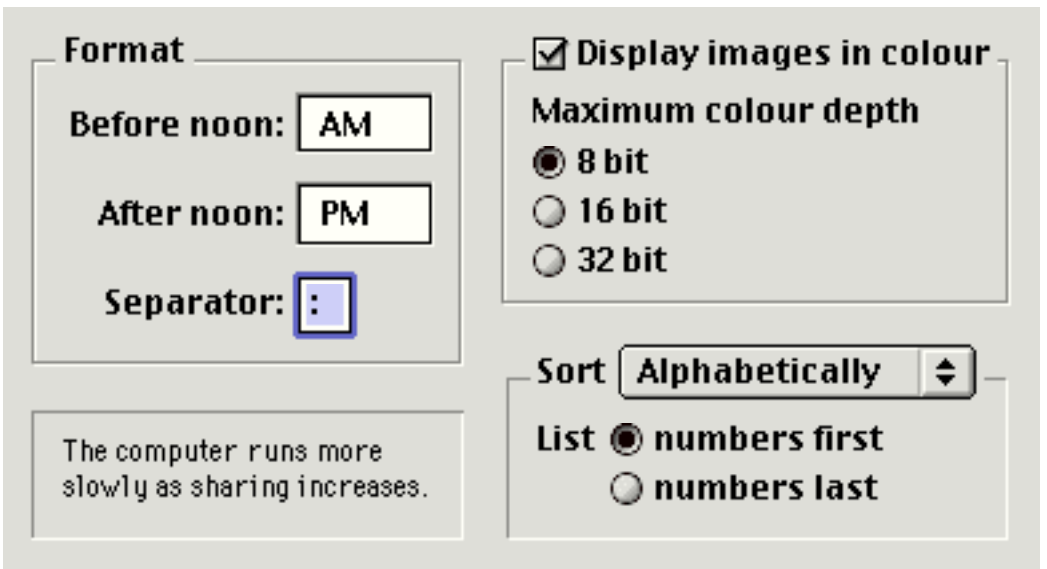

FIG 9 - TYPICAL GROUP BOXES

Variants and Control Definition IDs

The group box CDEF resource ID is 10. The six available variants and their control definition IDs are as follows:

Variant	Var Code	Control Definition ID
Primary group box, text title.	0	160 kControlGroupBoxTextTitleProc
Primary group box, checkbox title.	1	161 kControlGroupBoxCheckBoxProc
Primary group box, pop-up button title.	2	162 kControlGroupBoxPopupButtonProc
Secondary group box, text title.	4	164 kControlGroupBoxSecondaryTextTitleProc
Secondary group box, checkbox title.	5	165 kControlGroupBoxSecondaryCheckBoxProc
Secondary group box, pop-up button title.	6	166 kControlGroupBoxSecondaryPopupButtonProc

Control Values

Control Value	Content
Initial	Ignored if group box has text title. If the group box has a checkbox or pop-up button title, same value as the checkbox or pop-up button.
Minimum	Ignored if group box has text title. If the group box has a checkbox or pop-up button title, same minimum setting as the checkbox or pop-up button.
Maximum	Ignored if group box has text title. If the group box has a checkbox or pop-up button title, same maximum setting as the checkbox or pop-up button.

Control Data Tag Constant

Control Data Tag Constant	Meaning and Data Type Returned or Set
kControlGroupBoxMenuHandleTag	Gets the menu handle of a group box. **Data type returned or set:** MenuHandle
kControlGroupBoxFontStyleTag	Gets or sets the font style. **Data type returned or set:** ControlFontStyleRec

Control Part Codes

Constant	Value	Description
kControlNoPart	0	Returned if the group box title is a text title. If the group box title is a checkbox title or a pop-up menu button title, the user tracked completely out of the control.
kControlButtonPart	10	The group box title is a checkbox title and the check box was hit. The group box title is a pop-up menu button and the mouse button was released over the button.
kControlMenuPart	2	The group box title is a pop-up menu button and the mouse button was released in the menu.

Clock

Clock controls are a combination of an edit text control and little arrows. The user can change the date and time displayed by using the little arrows or by typing a value into the edit text control. Fig 10 shows a clock control displaying the date.

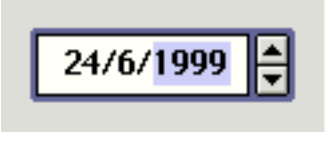

FIG 10 - CLOCK CONTROL DISPLAY

The clock control supports keyboard navigation and focus. When the clock control has keyboard focus, pressing the up-arrow and down-arrow keys has the same effect as clicking the up and down arrows in the control.

You can make a clock control inactive, in which case it displays time and date values without allowing the user to change them.

Variants and Control Definition IDs

The clock control CDEF resource ID is 15. The four available variants and their control definition ID are follows:

Variant	Var Code	Control Definition ID	
Clock control displaying hour/minutes.	0	240	kControlClockTimeProc
Clock control displaying hours/minutes/seconds.	1	241	kControlClockTimeSecondsProc
Clock control displaying date/month/year.	2	242	kControlClockDateProc
Clock control displaying month/year.	3	243	kControlClockMonthYearProc

Control Values

Control Value	Content
Initial	One or more of the clock value flags. (See Clock Value Flag Constants, below.) Reset to 0 after creation.
Minimum	Reserved. Set to 0.
Maximum	Reserved. Set to 0.

Clock Value Flag Constants

Constant	Value	Description
kControlClockNoFlags	0	Indicates that clock is editable but does not display the current "live" time.
kControlClockIsDisplayOnly	1	When only this bit is set, the clock is not editable. When this bit and the kControlClockIsLive bit is set, the clock automatically updates on idle (clock will have the current time).
kControlClockIsLive	2	When only this bit is set, the clock automatically updates on idle and any changes to the clock affect the system clock. When this bit and the kControlClockIsDisplayOnly bit is set, the clock automatically updates on idle (clock will have the current time), but is not editable.

Control Data Tag Constant

Control Data Tag Constant	Meaning and Data Type Returned or Set
kControlClockLongDateTag	Gets or sets the clock control's time or date. **Data type returned or set:** LongDateRec structure.
kControlClockFontStyleTag	Gets or sets the font style. **Data type returned or set:** ControlFontStyleRec

Control Part Codes

Constant	Value	Description
kControlClockPart	8	Event occurred in a clock control.
kControlClockHourDayPart	9	Event occurred in the part that contains the hour.
kControlClockMinuteMonthPart	10	Event occurred in the part that contains the minute or the month.
kControlClockSecondYearPart	11	Event occurred in the part that contains the second or the year.

kControlClockAMPMPart	12	Event occurred in the part that contains the AM/PM information.

Progress Indicators

Progress indicators are used to indicate that a lengthy operation is in progress. Two types of progress indicators can be used: an **indeterminate progress indicator**, which shows that an operation is occurring but does not indicate its duration; a **determinate progress indicator**, which shows how much of the operation has been completed. The two types are shown at Fig 11. Progress indicators are horizontal in orientation.

FIG 11 - PROGRESS INDICATORS

You might use a determinate progress indicator to, for example, show the progress of a file copying operation. You might use an indeterminate progress indicator to, for example, let the user know that the application is in the process of attempting a communication connection or is waiting to receive data during file transfer. You should consider switching from an indeterminate progress indicator to a determinate one if a process reaches a point where its scope becomes determinable. For example, a remote file transfer may not become determinate until the application has had time to establish a connection and calculate the data transfer rate.

Variants and Control Definition IDs

The progress indicator CDEF resource ID is 5. The one available variant and its control definition ID is follows:

Variant	Var Code	Control Definition ID
Progress indicator. To make the control determinate or indeterminate, set the kControlProgressBarIndeterminateTag constant. (See Control Data Tag Constant Relevant to Progress Indicators, below.)	0	80 kControlProgressBarProc

Progress Indicator Control Values

Control Value	Content
Initial	Appropriate value between –32768 and 32768.
Minimum	–32768 to 32768.
Maximum	–32768 to 32768.

Control Data Tag Constant

Control Data Tag Constant	Meaning and Data Type Returned or Set
kControlProgressBarIndeterminateTag	Gets or sets whether a progress indicator is determinate or indeterminate. **Data type returned or set:** Boolean. If true, switches to an indeterminate progress indicator. If false, switches to an determinate progress indicator.

Little Arrows

Little arrows provide the user with a means of increasing or decreasing values in a series, and consists of two arrows pointing in opposite directions. Little arrows should be accompanied by a label that identifies the content to which the control relates.

If the user clicks the up arrow, the value displayed is incremented by one unit of change. If the user presses the arrow, the value increases or decreases until the user releases the mouse button. While the user presses the arrow, it is highlighted to provide feedback to the user.

The unit of change should depend on the content. If the content area displays years, for example, the increment should be one year. On the other hand, if you used little arrows to control the size of a RAM cache, for example, you might use multiples of 32K as the increment.

Fig 12 shows a little arrows control used in conjunction with a placard and static text fields.

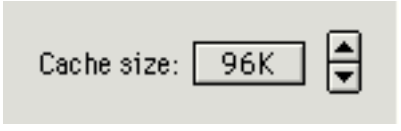

FIG 12 - LITTLE ARROWS WITH PLACARD AND STAT

Variant and Control Definition ID

The little arrows CDEF resource ID is 6. The one available variant and its control definition ID is as follows:

Variant	Var Code	Control Definition ID
Little arrows.	0	96 kControlLittleArrowsProc

Control Values

Control Value	Content
Initial	Appropriate value between –32768 and 32768.
Minimum	–32768 and 32768.
Maximum	–32768 and 32768.

Control Part Codes

Constant	Value	Description
kControlUpButtonPart	20	Event occurred in up arrow.
kControlDownButtonPart	21	Event occurred in down arrow.

Disclosure Triangles

Disclosure triangles are controls that allow the display or "disclosure" of information which elaborates on the primary information in a window. Disclosure triangles have two possible values: 0 for collapsed and 1 for expanded.

One usage of the disclosure triangle is to expand a dialog box or control panel. When the user clicks on the disclosure triangle, the triangle rotates downward and the window expands to provide additional information. A further click on the triangle rotates the triangle back to the right and restores the original appearance of the window.

Another usage pertains to list views (see Fig 13). The triangle appears next to the icon of each folder in the window. The user clicks the triangle to expand the view by revealing a list of the contents of the folder without opening it. The triangle rotates downward when the folder is expanded, which tells the user that the view is expanded even in cases when the folder is empty. Clicking the triangle again restores the view to its original (collapsed) state and turns the triangle back to the right.

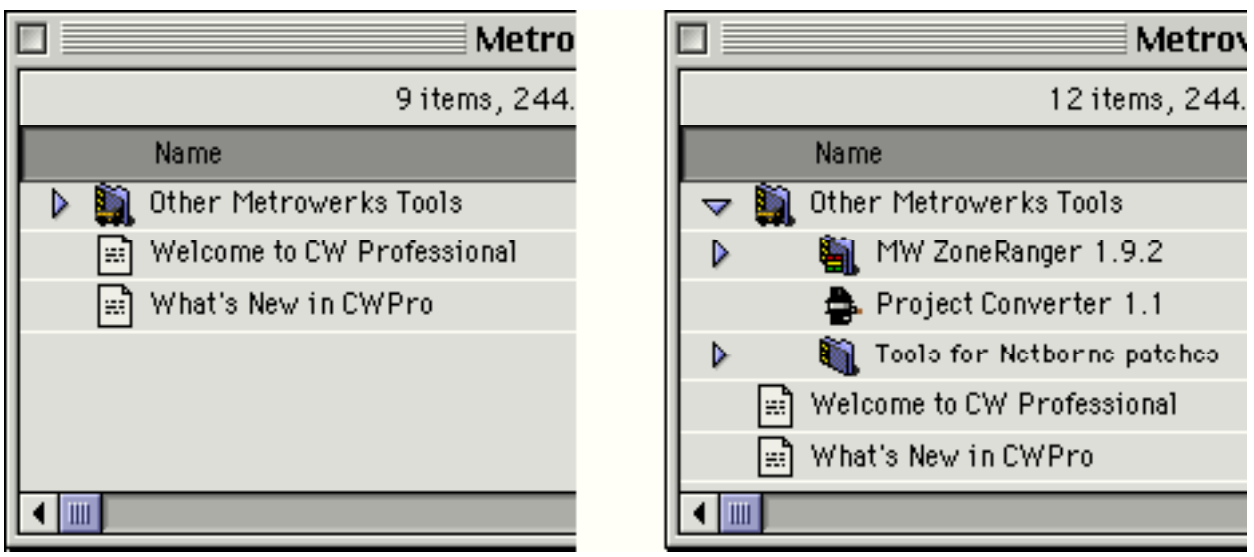

FIG 13 - DISCLOSURE TRIANGLES USED IN FINDER LIST VIEW

Variants and Control Definition IDs

The disclosure triangle CDEF resource ID is 4. The one available variant and its control definition ID is as follows:

Variant	Var Code	Control Definition ID
Disclosure triangle.	0	64 kControlTriangleProc
Left-facing disclosure triangle.	1	65 kControlTriangleLeftFacingProc
Auto-tracking disclosure triangle. This variant maintains its last value, so it knows what transition is taking place when a SetControlValue is called on it (expanded to collapsed, or vice versa). (See Control Data Tag Constants, below.)	2	66 kControlTriangleAutoToggleProc
Left-facing, auto-tracking disclosure triangle.	3	67 kControlTriangleLeftFacingAutoToggleProc

Control Values

Control Value	Content
Initial	0 (collapsed) or 1 (expanded)
Minimum	0 (collapsed)
Maximum	1 (expanded)

Control Data Tag Constant

Control Data Tag Constant	Meaning and Data Type Returned or Set
kControlTriangleLastValueTag	Gets or sets the last value of a disclosure triangle. Used primarily for setting up a disclosure triangle properly when using the kControlTriangleAutoToggleProc variant. **Data type returned or set:** SInt16

Control Part Codes

Constant	Value	Description
kControlTrianglePart	4	Event occurred in a disclosure triangle.

Picture

Picture controls display pictures.

Variants and Control Definition IDs

The picture control CDEF resource ID is 19. The two available variants and their control definition IDs are as follows:

Variant	Var Code	Control Definition ID	
Picture control.	0	304	kControlPictureProc
Non-tracking picture control. If hit, immediately returns the control part code kControlNoPart and does not track	1	305	kControlPictureNoTrackProc

Control Values

Control Value	Content
Initial	Resource ID of the 'PICT' resource you wish to display. Reset to 0 after creation.
Minimum	Reserved. Set to 0.
Maximum	Reserved. Set to 0.

Control Part Codes

Constant	Value	Description
kControlPicturePart	6	Event occurred in a picture control.

Icon

Icon controls display colour icons and icons from an icon suite.

A non-tracking variant is available for use in dialog boxes which have an embedding hierarchy and want an icon. This variant just returns the part hit immediately; it does not actually track the mouse.

Variants and Control Definition IDs

The icon CDEF resource ID is 20. The four available variants and their control definition IDs are as follows:

Variant	Var Code	Control Definition ID	
Icon control.	0	320	kControlIconProc
Non-tracking icon. If hit, immediately returns the control part code kControlIconPart without tracking.	1	321	kControlIconNoTrackProc
Icon suite.	2	322	kControlIconSuiteProc
Non-tracking icon suite. If hit, immediately returns the control part code kControlIconPart without tracking.	3	323	kControlIconSuiteNoTrackProc
Supports all standard types of icon-based content.	4	324	kControlIconRefProc

Supports all standard types of icon-based content. Non-tracking variant.	5	325 kControlIconRefNoTrackProc

Control Values

Control Value	Content
Initial	Resource ID of the 'cicn', 'ICON', or icon suite resource you wish to display. (The icon suite variant looks for an icon suite. If not found, it looks for a 'cicn' or 'ICON' resource.) Reset to 0 after creation.
Minimum	Reserved. Set to 0.
Maximum	Reserved. Set to 0.

Control Data Tag Constants

Control Data Tag Constant	Meaning and Data Type Returned or Set
kControlIconTransformTag	Gets or sets a transform that is added to the standard transform for an icon. (See Chapter 13 — Offscreen Graphics Worlds, Pictures, Cursors, and Icons.) **Data type returned or set:** IconTransformType
kControlIconAlignmentTag	Gets or sets an icon's position. (See Chapter 13 — Offscreen Graphics Worlds, Pictures, Cursors, and Icons.) **Data type returned or set:** IconAlignmentType
kControlIconResourceIDTag	Gets or sets the resource ID of the icon to use. **Data type returned or set:** Sint16
kControlIconContentTag	Gets or sets the type of content to be used in an icon control. **Data type returned or set:** ControlButtonContentInfo

Control Part Codes

Constant	Value	Description
kControlIconPart	7	Event occurred in an icon control.

Window Headers

Window headers (see Fig 14) are bevelled rectangles which should be located at the top of a window's content region and used to provide information about the window's contents. The outside lines should share the same space as the inside lines of the document window and the scroll bar arrows.

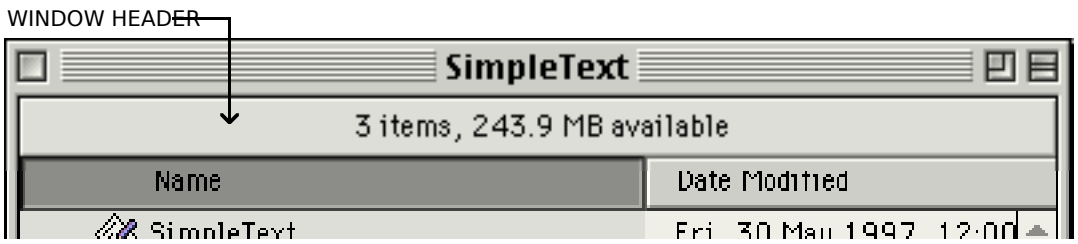

FIG 14 - WINDOW HEADER

The list view header variant provides the same functionality as the window header, but removes the line which separates a standard window header from the content area.

Variants and Control Definition IDs

The window header CDEF resource ID is 21. The two available variants and their control definition IDs are as follows:

Variant	Var Code	Control Definition ID

Window header.	0	336	kControlWindowHeaderProc
Window list view header.	1	337	kControlWindowListViewHeaderProc

Control Values

Control Value	Content
Initial	Reserved. Set to 0.
Minimum	Reserved. Set to 0.
Maximum	Reserved. Set to 0.

Placard — Embedding Control

Placards, which are embedding controls, are rectangular controls used as an information display or as background fill for a control area. Typically, placards are used as a small information panel placed at the bottom of a window to the left of the horizontal scroll bar (see Fig 15).

FIG 15 - PLACARD

Variants and Control Definition IDs

The placard CDEF resource ID is 14. The one available variant and its control definition ID is as follows:

Variant	Var Code	Control Definition ID
Placard.	0	224 kControlPlacardProc

Future versions will provide a push button variant.

Control Values

Control Value	Content
Initial	Reserved. Set to 0.
Minimum	Reserved. Set to 0.
Maximum	Reserved. Set to 0.

Static Text

Static text controls display static text, that is, text that cannot be changed by the user.

Variant and Control Definition ID

The static text control CDEF resource ID is 18. The one available variant and its control definition ID is as follows:

Variant	Var Code	Control Definition ID
Static text control.	0	288 kControlStaticTextProc

Control Values

Control Value	Content
Initial	Reserved. Set to 0.
Minimum	Reserved. Set to 0.

Maximum	Reserved. Set to 0.

Control Data Tag Constants

Control Data Tag Constant	Meaning and Data Type Returned or Set
kControlStaticTextTextTag	Gets or sets text in a static text control. **Data type returned or set:** character buffer
kControlStaticTextTextHeightTag	Gets the height of text in a static text control. **Data type returned or set:** SInt16
kControlStaticTextStyleTag	Gets or sets the font style. **Data type returned or set:** ControlFontStyleRec
kControlStaticTextTruncTag	Gets or or sets how text is truncated at the end of a line for a static text control. **Data type returned or set:** TruncCode. **The value** truncEnd **indicates that characters are truncated off the end of the string**. truncMiddle **indicates that characters are truncated from the middle of the string. The default is** -1**, which indicates that no truncation occurs and, instead, the text is wrapped**.

Separator Lines

Separator lines (see Fig 16) are two-pixel-wide vertical or horizontal lines used to visually separate groups of controls. The orientation of the bounding rectangle determines the orientation of the line.

FIG 16 - SEPARATOR LINE

Variants and Control Definition IDs

The separator line CDEF resource ID is 9. The one available variant and its control definition ID is as follows:

Variant	Var Code	Control Definition ID
Separator line.	0	144 kControlSeparatorLineProc

Control Values

Control Value	Content
Initial	Reserved. Set to 0.
Minimum	Reserved. Set to 0.
Maximum	Reserved. Set to 0.

Pop-up Arrows

The pop-up arrow control simply draws the pop-up glyph.

Variants and Control Definition IDs

The pop-up arrow CDEF resource ID is 12. The eight available variants and their control definition IDs are as follows:

Variant	Var Code	Control Definition ID
Large, right-facing pop-up arrow.	0	192 kControlPopupArrowEastProc

Large, left-facing pop-up arrow.	1	193	kControlPopupArrowWestProc
Large, up-facing pop-up arrow.	2	194	kControlPopupArrowNorthProc
Large, down-facing pop-up arrow.	3	195	kControlPopupArrowSouthProc
Small, right-facing pop-up arrow.	4	196	kControlPopupArrowSmallEastProc
Small, left-facing pop-up arrow.	5	197	kControlPopupArrowSmallWestProc
Small, up-facing pop-up arrow.	6	198	kControlPopupArrowSmallNorthProc
Small, down-facing pop-up arrow.	7	198	kControlPopupArrowSmallSouthProc

Control Values

Control Value	Content
Initial	Reserved. Set to 0.
Minimum	Reserved. Set to 0..
Maximum	Reserved. Set to 0.

Radio Groups — Embedding Control

Radio groups are embedder controls which relieve your application of much of the work involved in managing a group of radio buttons (or bevel buttons which are intended to operate like radio buttons). For example, if a group of radio buttons are embedded in a radio group control, the radio group control handles the checking and unchecking of the radio buttons when the user clicks on one of them. The current value of the radio group control represents the radio button currently selected.

Variant and Control Definition ID

The radio group CDEF resource ID is 26. The one available variant and its control definition ID is as follows:

Variant	Var Code	Control Definition ID	
Radio group.	0	416	kControlRadioGroupProc

Control Values

Control Value	Content
Initial	Set to 0 on creation. Reset to the index of currently selected embedded radio control after creation. If the currently selected control does not support radio behaviour, this value will be set to 0 and the control will be deselected. To deselect all controls, set to 0.
Minimum	Set to 0.
Maximum	Set to 0 on creation. Reset to the number of embedded controls as controls are added.

Control Part Codes

Constant	Value	Description
kControlRadioGroupPart	27	Event occurred in a radio group.

Asynchronous Arrows

Asynchronous arrows (see Fig 17) are a simple animation used to indicate that an asynchronous background process is occurring, in other words a process which does not display a dialog box containing a progress indicator. Asynchronous arrows are also known as "chasing arrows".

FIG 17 - FRAMES OF ASYNCHRONOUS ARROWS ANIM

Variant and Control Definition ID

The asynchronous arrows CDEF resource ID is 7. The one available variant and its control definition ID is as follows:

Variant	Var Code	Control Definition ID
Asynchronous arrows.	0	112 kControlChasingArrowsProc

Control Values

Control Value	Content
Initial	Reserved. Set to 0.
Minimum	Reserved. Set to 0..
Maximum	Reserved. Set to 0.

User Panes — Embedding Control

The user pane has two main uses:

- It can be used as an embedder control, that is, other controls may be embedded within it. It is thus particularly useful for grouping together the controls belonging to an individual pane of a tab control or pop-up button group box.
- It provides a way to hook in application-defined functions, known as **user pane functions** (see below), which perform actions such as drawing, hit testing, tracking, etc.

Variants and Control Definition IDs

The user pane CDEF resource ID is 16. The one available variant and its control definition ID is as follows:

Variant	Var Code	Control Definition ID
User pane.	0	256 kControlUserPaneProc

Control Values

Control Value	Content
Initial	One or more of the control feature constants. (See Defining Your Own User Pane Functions, below.) Reset to 0 after creation.
Minimum	Ignored. After user pane creation, reset to a setting between –32768 to 32768.
Maximum	Ignored. After user pane creation, reset to a setting between –32768 to 32768.

Control Data Tag Constants

The control data tag constants relevant to user panes all relate to user pane functions. See Defining Your Own User Pane Function, below.

Scrolling Text Box

The scrolling text box implements a scrolling box of non-editable text. This is useful for credits in About boxes, etc. There are two variants:

- The standard variant, which has a scroll bar.
- The auto-scrolling variant, which does not have a scroll bar. This variant needs two pieces of information to work: the delay (in ticks) before the scrolling starts; the time (in ticks) between scrolls. By default, the text will scroll one pixel at a time, although this may be changed via SetControlData.

In order to advance the scrolling in the auto-scrolling variant, you must call IdleControls on a periodic basis, such as when you receive a null event.

Variants and Control Definition IDs

The scrolling text box CDEF resource ID is27. The two available variants and their control definition IDs are as follows:

Variant	Var Code	Control Definition ID
Standard.	0	432 kControlScrollTextBoxProc
Auto-scrolling.	1	433 kControlScrollTextBoxAutoScrollProc

Control Values

Control Value	Content
Initial	Resource ID of a 'TEXT' and, optionally, a 'styl' resource.
Minimum	For the standard variant, set to 0. For the auto-scrolling variant , the delay (in ticks) before scrolling begins. (This delay is also used between when the scrolling completes and when it begins again.)
Maximum	For the standard variant, set to 0. For the auto-scrolling variant, the delay (in ticks) between each unit of scrolling.

Control Data Tag Constants

Control Data Tag Constant	Meaning and Data Type Returned or Set
KControlScrollTextBoxDelayBeforeAutoScrollTag	Gets or sets the number of ticks of delay before the initial scrolling of an auto-scrolling text box begins. **Data type returned or set:** UInt32
KControlScrollTextBoxDelayBetweenAutoScrollTag	Gets or sets the number of ticks of delay between each unit of scrolling in an auto-scrolling text box. **Data type returned or set:** UInt32
KControlScrollTextBoxAutoScrollAmountTag	Gets or sets the number of pixels by which an auto scrolling text box scrolls. (The default is 1.) **Data type returned or set:** UInt16
kControlScrollTextBoxContentsTag	Sets the ID of a 'TEXT' resource and, optionally, a 'styl' resource, to be used as the content of a standard or auto-scrolling text box. **Data set:** Sint16

Idle Processing

The following four controls need to perform idle processing for the reasons indicated:

Control	Reason For Idle Processing

Progress Indicator (indeterminate variant).	Animate the indicator.
Clocks (when the kControlClockIsLive clock value flag constant is used).	Update the clock.
Asynchronous arrows.	Animate the arrows.
Edit text controls.	Cause TextEdit to be called to blink the insertion point caret.

When these controls are displayed in a document window, your application should call the IdleControls function at least once in your event loop. IdleControls sends a particular message to the CDEF, which responds appropriately.

Defining Your Own Key Filter Function

Controls that support text input (such as edit text control and list box controls) can attach an application-defined **key filter function** to filter key strokes or modify them on return. . Your key filter function can change the keystroke, leave it alone, or block the CDEF from receiving it. For example, an edit text control could use a key filter function to allow only numeric values to be input in its field.

The Control Manager declares the type for an application-defined key filter function as follows:

```
ControlKeyFilterProcPtr = FUNCTION(theControl: ControlHandle; VAR keyCode: SInt16;
                        VAR charCode: SInt16; VAR modifiers: SInt16): ControlKeyFilterResult;
```

The Control Manager defines the data type ControlKeyFilterUPP to identify the universal procedure pointer for this application-defined function:

```
ControlKeyFilterUPP = UniversalProcPtr;
```

As stated at Edit Text Fields, above, the control data tag constant for getting and setting a key filter function is kControlKeyFilterTag and the data type returned or set is ControlKeyFilterUPP.

Example

The following example relates to an edit text field and assumes an application-defined key filter function named numericFilter.

```
Const
kLeftArrow = $1C;
kRightArrow = $1D;
kUpArrow = $1E;
kDownArrow = $1F;
kBackspace = $08;
...

numericFilterUPP : ControlKeyFilterUPP;
controlHdl : ControlHandle;
...

//
.....
create routine descriptor

numericFilterUPP := NewControlKeyFilterProc(@NumericFilter);

// ..... attach key filter function to
the control

ignoredErr := GetDialogItemAsControl(dialogPtr, itemNumber, controlHdl);
ignoredErr := SetControlData(controlHdl, kControlNoPart,
   kControlEditTextKeyFilterTag, sizeof(numericFilterUPP), @NumericFilterUPP);

...
```

```
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ NumericFilter
//
// This function will be called each time the edit text field receives a keystroke.
// Keystrokes that are to be accepted must return kControlKeyFilterPassKey.  Keystrokes
// that are to be blocked must return kControlKeyFilterBlockKey.  This function blocks
// all but the numeric keys, the "dash" key, and the arrow keys.

function NumericFilter(theControl: ControlHandle; VAR keyCode: SInt16;
        VAR charCode: SInt16; VAR modifiers: SInt16): ControlKeyFilterResult;

begin
if ((char(charCode) >= '0') and (char(charCode) <= '9')) then
    begin
    NumericFilter := kControlKeyFilterPassKey;
    Exit(NumericFilter);
    end;

case char(charCode) of

    '-', kLeftArrow, kRightArrow, kUpArrow, kDownArrow, kBackspace: begin
        NumericFilter := kControlKeyFilterPassKey;
        Exit(NumericFilter);
        end;

    otherwise begin
        SysBeep(10);
        NumericFilter := kControlKeyFilterBlockKey;
        Exit(NumericFilter);
        end;;
    end;
        { of case statement }
end;
```

Defining Your Own Edit Text Validation Function

A key filter function, however, does not cater for the case of pasting text to an edit text item. Accordingly, you will ordinarily want to combine an **edit text validation function** with the key filter function for a specific edit text control. The following example ensures that, if a user supplied filename pasted to the edit text item contains one or more colons, those colons will be replaced with dashes.

```
procedure editTextValidator (controlHdl : ControlHandle);

    var
    theText : Str31;
    actualSize: Size;
    a : UInt8;

    { Get the text to be examined from the control }

    GetControlData(controlHdl, kControlNoPart, kControlEditTextTextTag,
                            sizeof(theText) -1, SInt16(theText[0]), actualSize);

    { Set the length byte to the number of characters in the text, limited to the current
        maximum for filenames }

    if (actualSize <= 31) then
        begin
        SInt16(theText[0]) := actualSize;
        end
    else begin
        SInt16(theText[0]) := 31;
        end;

    { Replace any colons with dashes }

    for a := 1 to SInt16(theText[0]) do
        begin
        if (theText[a] = ':') then
            begin
            theText[a] := '-';
            end;
        end;
```

```
{ You might want to add code here to check whether any characters were replaced before
   bothering to redraw }

{ Put the replaced text into the control and redraw the control }

SetControlData(controlHdl, kControlNoPart, kControlEditTextTextTag,
                        SInt16(theText[0]), @theText[1]);

Draw1Control(controlHdl);
end;
   { of procedure editTextValidator }
```

Defining Your Own User Pane Functions

As previously stated, one of the functions of a user pane is to provide a way to hook in application-defined functions which perform actions such as drawing, hit testing, tracking, etc. Such application-defined functions are called **user pane functions**.

User pane functions provide you with the ability to create a custom Appearance-compliant control without writing your own control definition function.

Once you have provided a user pane function, you call SetControlData with the control data tag constant representing the user pane function you wish to get or set passed in the inTagName parameter and a universal procedure pointer to the user pane function passed in the inData parameter.

User Pane Functions

User pane functions are categorised as follows:

Function	Description
Draw	Draws the content of a user pane control in the rectangle of user pane control.
Hit test	Returns the part code of the control that the point was in when the mouse-down event occurred.
Tracking	Tracks a control while the user holds down the mouse button. The function should track the control by repeatedly calling the action function specified in the actionProc parameter until the mouse button is released. When the mouse button is released, your function should return the part code of the control part that was tracked. This function will only get called if you have set the kControlHandlesTracking control feature bit on creation of the user pane control.
Idle	Performs idle processing. This function will only get called if you have set the kControlWantsIdle control feature bit on creation of the user pane control.
Key Down	Handles keyboard event processing. The function should handle the key pressed or released by the user and return the part code of the control where the keyboard event occurred. This function will only get called if you've set the kControlSupportsFocus control feature bit on creation of the user pane control.
Activate	Handles activate and deactivate event processing. The function should perform any special processing before the user pane becomes activated or deactivated. For example, it should deactivate its TEHandle or ListHandle if the user pane is about to be deactivated. This function will only get called if you have set the kControlWantsActivate control feature bit on creation of the user pane control.
Focus	Handle keyboard focus. The function is called in response to a change in keyboard focus. It should respond by changing keyboard focus based on the part code passed in the action parameter. This function will only get called if you have set the kControlSupportsFocus control feature bit on creation of the user pane control.

Background	Sets the background colour or pattern (only for user panes that support embedding). The function should set the user pane background colour or pattern to whatever is appropriate given the bit depth and device type passed in. This function will only get called if you have set the kControlHasSpecialBackground and kControlSupportsEmbedding control feature bits on creation of the user pane control.

Control Feature Flags

As stated at User Panes, above, the initial value of a user pane is one or more of the control feature constants. As stated in the preceding list of user pane functions, certain user pane functions will only get called if certain control feature flags are set. The control feature flags relevant to user pane functions are as follows:

```
KControlSupportsEmbedding       = $02;
KControlSupportsFocus           = $04;
KControlWantsIdle               = $08;
KControlWantsActivate           = $10;
KControlHandlesTracking         = $20;
KControlHasSpecialBackground    = $80;
```

Type Definitions

The Control Manager declares the following types for user pane functions:

```
ControlUserPaneDrawProcPtr = PROCEDURE(control: ControlHandle; part: SInt16);

ControlUserPaneHitTestProcPtr = FUNCTION(control: ControlHandle; where: Point) :
                                ControlPartCode;

ControlUserPaneTrackingProcPtr = FUNCTION(control: ControlHandle; startPt: Point;
                                actionProc: ControlActionUPP): ControlPartCode;

ControlUserPaneIdleProcPtr = PROCEDURE(control: ControlHandle);

ControlUserPaneKeyDownProcPtr = FUNCTION(control: ControlHandle; keyCode: SInt16;
                                charCode: SInt16; modifiers: SInt16): ControlPartCode;

ControlUserPaneActivateProcPtr = PROCEDURE(control: ControlHandle; activating: BOOLEAN);

ControlUserPaneFocusProcPtr = FUNCTION(control: ControlHandle;
                                action: ControlFocusPart): ControlPartCode;

ControlUserPaneBackgroundProcPtr = PROCEDURE(control: ControlHandle;
                                info: ControlBackgroundPtr);
```

The Control Manager defines the following data types to identify the universal procedure pointer for user pane functions:

```
ControlUserPaneDrawUPP = UniversalProcPtr;
ControlUserPaneHitTestUPP = UniversalProcPtr;
ControlUserPaneTrackingUPP = UniversalProcPtr;
ControlUserPaneIdleUPP = UniversalProcPtr;
ControlUserPaneKeyDownUPP = UniversalProcPtr;
ControlUserPaneActivateUPP = UniversalProcPtr;
ControlUserPaneFocusUPP = UniversalProcPtr;
ControlUserPaneBackgroundUPP = UniversalProcPtr;
```

Control Data Tag Constants

The control data tag constants relating to user pane functions are as follows:

Control Data Tag Constant	Meaning and Data Type Returned or Set
kControlUserPaneDrawProcTag	Gets or sets a user pane drawing function. Indicates that the Control Manager needs to draw a control. **Data type returned or set:** ControlUserPaneDrawingUPP
kControlUserPaneHitTestProcTag	Gets or sets a user pane hit-testing function. Indicates that the Control Manager needs to determine if a control part was hit. **Data type returned or set:** ControlUserPaneHitTestUPP

kControlUserPaneTrackingProcTag	Gets or sets a user pane tracking function, which will be called when a control definition function returns the kControlHandlesTracking feature bit in response to a kControlMsgGetFeatures message. Indicates that a user pane handles its own tracking. **Data type returned or set:** ControlUserPaneTrackingUPP
kControlUserPaneIdleProcTag	Gets or sets a user pane idle function, which will be called when a control definition function returns the kControlWantsIdle feature bit in response to kControlMsgGetFeatures message. Indicates that a user pane performs idle processing. **Data type returned or set:** ControlUserPaneIdleUPP
kControlUserPaneKeyDownProcTag	Gets or sets a user pane key down function, which will be called when a control definition function returns the kControlSupportsFocus feature bit in response to a kControlMsgGetFeatures message. Indicates that a user pane performs keyboard event processing. **Data type returned or set:** ControlUserPaneKeyDownUPP
kControlUserPaneActivateProcTag	Gets or sets a user pane activate function, which will be called when a control definition function returns the kControlWantsActivate feature bit in response to a kControlMsgGetFeatures message. Indicates that a user pane wants to be informed of activate and deactivate events. **Data type returned or set:** ControlUserPaneActivateUPP
kControlUserPaneFocusProcTag	Gets or sets a user pane keyboard focus function, which will be called when a control definition function returns the kControlSupportsFocus feature bit in response to a kControlMsgGetFeatures message. Indicates that a user pane handles keyboard focus. **Data type returned or set:** ControlUserPaneFocusUPP
kControlUserPaneBackgroundProcTag	Gets or sets a user pane background function, which will be called when a control definition function returns the kControlHasSpecialBackground and kControlSupportsEmbedding feature bits in response to a kControlMsgGetFeatures message. Indicates that a user pane can set its background colour or pattern. **Data type returned or set:** ControlUserPaneBackgroundUPP

Main Constants, Data Types and Functions

All of the following constants, data types, and functions, with the exception of some of the control part code constants, were introduced with Mac OS 8 and the Appearance Manager. Those introduced with Mac OS 8.5 appear on a dark gray background.

Constants

Control Definition IDs

```
KControlBevelButtonSmallBevelProc              = 32
KControlBevelButtonNormalBevelProc             = 33
KControlBevelButtonLargeBevelProc              = 34
kControlSliderProc                             = 48
kControlSliderLiveFeedback                     = $01
kControlSliderHasTickMarks                     = $02
kControlSliderReverseDirection                 = $04
kControlSliderNonDirectional                   = $08
kControlTriangleProc                           = 64
kControlTriangleLeftFacingProc                 = 65
kControlTriangleAutoToggleProc                 = 66
kControlTriangleLeftFacingAutoToggleProc       = 67
kControlProgressBarProc                        = 80
kControlLittleArrowsProc                       = 96
kControlChasingArrowsProc                      = 112
```

kControlTabLargeProc	= 128
kControlTabSmallProc	= 129
kControlSeparatorLineProc	= 144
kControlGroupBoxTextTitleProc	= 160
kControlGroupBoxCheckBoxProc	= 161
kControlGroupBoxPopupButtonProc	= 162
kControlGroupBoxSecondaryTextTitleProc	= 164
kControlGroupBoxSecondaryCheckBoxProc	= 165
kControlGroupBoxSecondaryPopupButtonProc	= 166
kControlImageWellProc	= 176
kControlPopupArrowEastProc	= 192
kControlPopupArrowWestProc	= 193
kControlPopupArrowNorthProc	= 194
kControlPopupArrowSouthProc	= 195
kControlPopupArrowSmallEastProc	= 196
kControlPopupArrowSmallWestProc	= 197
kControlPopupArrowSmallNorthProc	= 198
kControlPopupArrowSmallSouthProc	= 199
kControlPlacardProc	= 224
kControlClockTimeProc	= 240
kControlClockTimeSecondsProc	= 241
kControlClockDateProc	= 242
kControlClockMonthYearProc	= 243
kControlUserPaneProc	= 256
kControlEditTextProc	= 272
kControlEditTextDialogProc	= 273
kControlEditTextPasswordProc	= 274
kControlEditTextInlineInputProc	= 276
kControlStaticTextProc	= 288
kControlPictureProc	= 304
kControlPictureNoTrackProc	= 305
kControlIconProc	= 320
kControlIconNoTrackProc	= 321
kControlIconSuiteProc	= 322
kControlIconSuiteNoTrackProc	= 323
kControlIconRefProc	= 324
kControlIconRefNoTrackProc	= 325
kControlWindowHeaderProc	= 336
kControlWindowListViewHeaderProc	= 337
kControlListBoxProc	= 352
kControlListBoxAutoSizeProc	= 353
kControlRadioGroupProc	= 416
kControlScrollTextBoxProc	= 432
kControlScrollTextBoxAutoScrollProc	= 433

Control Variants

kControlNoVariant	= 0
kControlUsesOwningWindowsFontVariant	= $08

Control Part Codes

kControlNoPart	= 0
kControlLabelPart	= 1
kControlMenuPart	= 2
kControlTrianglePart	= 4
kControlEditTextPart	= 5
kControlPicturePart	= 6
kControlIconPart	= 7
kControlClockPart	= 8
kControlClockHourDayPart	= 9,
kControlClockMinuteMonthPart	= 10
kControlClockSecondYearPart	= 11
kControlClockAMPMPart	= 12
kControlListBoxPart	= 24
kControlListBoxDoubleClickPart	= 25
kControlImageWellPart	= 26
kControlRadioGroupPart	= 27
kControlButtonPart	= 10
kControlCheckBoxPart	= 11
kControlRadioButtonPart	= 11
kControlUpButtonPart	= 20
kControlDownButtonPart	= 21
kControlPageUpPart	= 22
kControlPageDownPart	= 23
kControlIndicatorPart	= 129
kControlDisabledPart	= 254
kControlInactivePart	= 255

Bevel Button Graphic Alignment

```
KControlBevelButtonAlignSysDirection        = -1
KControlBevelButtonAlignCenter              = 0
KControlBevelButtonAlignLeft                = 1
KControlBevelButtonAlignRight               = 2
KControlBevelButtonAlignTop                 = 3
kControlBevelButtonAlignBottom              = 4
kControlBevelButtonAlignTopLeft             = 5
kControlBevelButtonAlignBottomLeft          = 6
kControlBevelButtonAlignTopRight            = 7
kControlBevelButtonAlignBottomRight         = 8
```

Bevel Button Text Alignment values

```
KControlBevelButtonAlignTextSysDirection    = teFlushDefault
KControlBevelButtonAlignTextCenter          = teCenter
KControlBevelButtonAlignTextFlushRight      = teFlushRight
KControlBevelButtonAlignTextFlushLeft       = teFlushLeft
```

Bevel Button Text Placement

```
KControlBevelButtonPlaceSysDirection        = -1
KControlBevelButtonPlaceNormally            = 0
KControlBevelButtonPlaceToRightOfGraphic    = 1
KControlBevelButtonPlaceToLeftOfGraphic     = 2
KControlBevelButtonPlaceBelowGraphic        = 3
KControlBevelButtonPlaceAboveGraphic        = 4
```

Bevel Button Behaviour

```
kControlBehaviorPushbutton                  = 0
kControlBehaviorToggles                     = $0100
kControlBehaviorSticky                      = $0200
kControlBehaviorMultiValueMenu              = $4000
kControlBehaviorOffsetContents              = $8000
kControlBehaviorCommandMenu                 = $2000
```

Bevel Button Content Type

```
kControlContentTextOnly                     = 0
kControlContentIconSuiteRes                 = 1
kControlContentCIconRes                     = 2
kControlContentPictRes                      = 3
kControlContentIconSuiteHandle              = 129
kControlContentCIconHandle                  = 130
kControlContentPictHandle                   = 131
kControlContentIconRef                      = 132
```

Clock Value Flag Constants

```
kControlClockNoFlags                        = 0
kControlClockIsDisplayOnly                  = 1
kControlClockIsLive                         = 2
```

Control Data Tags

```
kControlFontStyleTag                        = 'font'
kControlKeyFilterTag                        = 'fltr'
kControlBevelButtonContentTag               = 'cont'
kControlBevelButtonTransformTag             = 'tran'
kControlBevelButtonTextAlignTag             = 'tali'
kControlBevelButtonTextOffsetTag            = 'toff'
kControlBevelButtonGraphicAlignTag          = 'gali'
kControlBevelButtonGraphicOffsetTag         = 'goff'
kControlBevelButtonTextPlaceTag             = 'tplc'
kControlBevelButtonMenuValueTag             = 'mval'
kControlBevelButtonMenuHandleTag            = 'mhnd'
kControlBevelButtonCenterPopupGlyphTag      = 'pglc'
kControlBevelButtonLastMenuTag              = 'lmnu'
kControlBevelButtonMenuDelayTag             = 'mdly'
kControlBevelButtonScaleIconTag                     = FOUR_CHAR_CODE('scal')
kControlTriangleLastValueTag                = 'last'
kControlProgressBarIndeterminateTag         = 'inde'
kControlTabContentRectTag                   = 'rect'
kControlTabEnabledFlagTag                   = 'enab'
```

```
kControlTabFontStyleTag                     = kControlFontStyleTag
kControlGroupBoxMenuHandleTag               = 'mhan'
kControlGroupBoxFontStyleTag                = kControlFontStyleTag
kControlImageWellContentTag                 = 'cont'
kControlImageWellTransformTag               = 'tran'
kControlClockLongDateTag                    = 'date'
kControlClockFontStyleTag                   = kControlFontStyleTag
kControlUserItemDrawProcTag                 = 'uidp'
kControlUserPaneDrawProcTag                 = 'draw'
kControlUserPaneHitTestProcTag              = 'hitt'
kControlUserPaneTrackingProcTag             = 'trak'
kControlUserPaneIdleProcTag                 = 'idle'
kControlUserPaneKeyDownProcTag              = 'keyd'
kControlUserPaneActivateProcTag             = 'acti'
kControlUserPaneFocusProcTag                = 'foci'
kControlUserPaneBackgroundProcTag           = 'back'
kControlEditTextStyleTag                    = kControlFontStyleTag
kControlEditTextTextTag                     = 'text'
kControlEditTextTEHandleTag                 = 'than'
kControlEditTextKeyFilterTag                = kControlKeyFilterTag,
kControlEditTextSelectionTag                = 'sele'
kControlEditTextPasswordTag                 = 'pass'
kControlEditTextLockedTag                           = FOUR_CHAR_CODE('lock')
kControlEditTextValidationProcTag                   = FOUR_CHAR_CODE('vali')
kControlStaticTextStyleTag                          = kControlFontStyleTag
kControlStaticTextTextTag                           = FOUR_CHAR_CODE('text')
kControlStaticTextTextHeightTag                     = FOUR_CHAR_CODE('thei')
kControlStaticTextTruncTag                          = FOUR_CHAR_CODE('trun')
kControlIconTransformTag                            = FOUR_CHAR_CODE('trfm')
kControlIconAlignmentTag                            = FOUR_CHAR_CODE('algn')
kControlIconResourceIDTag                           = FOUR_CHAR_CODE('ires')
kControlIconContentTag                              = FOUR_CHAR_CODE('cont')
kControlListBoxListHandleTag                        = FOUR_CHAR_CODE('lhan')
kControlListBoxKeyFilterTag                         = kControlKeyFilterTag
kControlListBoxFontStyleTag                         = kControlFontStyleTag
kControlListBoxDoubleClickTag                       = FOUR_CHAR_CODE('dblc')
kControlScrollTextBoxDelayBeforeAutoScrollTag       = FOUR_CHAR_CODE('stdl')
kControlScrollTextBoxDelayBetweenAutoScrollTag      = FOUR_CHAR_CODE('scdl')
kControlScrollTextBoxAutoScrollAmountTag            = FOUR_CHAR_CODE('samt')
kControlScrollTextBoxContentsTag                    = FOUR_CHAR_CODE('tres')
```

Key Filter Result Codes

```
kControlKeyFilterBlockKey                   = 0
kControlKeyFilterPassKey                    = 1
```

Control Feature Bits

```
kControlSupportsGhosting                    = $0001
kControlSupportsEmbedding                   = $0002
kControlSupportsFocus                       = $0004
kControlWantsIdle                           = $0008
kControlWantsActivate                       = $0010
kControlHandlesTracking                     = $0020
kControlSupportsDataAccess                  = $0040
kControlHasSpecialBackground                = $0080
kControlGetsFocusOnClick                    = $0100
kControlSupportsCalcBestRect                = $0200
kControlSupportsLiveFeedback                = $0400
kControlHasRadioBehavior                    = $0800
```

Focusing Part Codes

```
KcontrolFocusNoPart                         = 0
KControlFocusNextPart                       = -1
kControlFocusPrevPart                       = -2
```

Data Types

```
ControlFocusPart : SInt16;
ControlKeyFilterResult : SInt16;
ControlButtonGraphicAlignment : SInt16;
ControlButtonTextAlignment : SInt16;
ControlButtonTextPlacement : SInt16;
ControlContentType : SInt16;
ControlVariant : SInt16;
```

Bevel Button and Image Well Content Structure

```
ControlButtonContentInfoPtr = ^ControlButtonContentInfo;
ControlButtonContentInfo = RECORD
    contentType : ControlContentType;
    CASE INTEGER OF
            0: (
                    resID:   SInt16;
                    );
            1: (
                    cIconHandle: CIconHandle;
                    );
            2: (
                    iconSuite: Handle;
                    );
            3: (
                    iconRef:    Handle;
                    );
            4: (
                    picture: PicHandle;
                    );
    END;
```

Edit Text Selection Structure

```
ControlEditTextSelectionRecPtr = ^ControlEditTextSelectionRec;
ControlEditTextSelectionRec = RECORD
    { Structure for getting the edit text selection. }
    selStart:          SInt16;
    selEnd:            SInt16;
    END;

ControlEditTextSelectionPtr = ^ControlEditTextSelectionRec;
```

Tab Information Structure

```
ControlTabInfoRecPtr = ^ControlTabInfoRec;
ControlTabInfoRec = RECORD
    version:                SInt16;         { Version of this structure. }
    iconSuiteID:            SInt16;         { Icon suite to use. Zero indicates no icon. }
    name:                   Str255;         { Name to be displayed on the tab. }
    END;
```

User Pane Functions

```
ControlUserPaneDrawProcPtr = PROCEDURE(control: ControlHandle; part: SInt16);

ControlUserPaneHitTestProcPtr = FUNCTION(control: ControlHandle; where: Point) :
    ControlPartCode;

ControlUserPaneTrackingProcPtr = FUNCTION(control: ControlHandle; startPt: Point;
    actionProc: ControlActionUPP): ControlPartCode;

ControlUserPaneIdleProcPtr = PROCEDURE(control: ControlHandle);

ControlUserPaneKeyDownProcPtr = FUNCTION(control: ControlHandle; keyCode: SInt16;
    charCode: SInt16; modifiers: SInt16): ControlPartCode;

ControlUserPaneActivateProcPtr = PROCEDURE(control: ControlHandle; activating: BOOLEAN);

ControlUserPaneFocusProcPtr = FUNCTION(control: ControlHandle;
    action: ControlFocusPart): ControlPartCode;

ControlUserPaneBackgroundProcPtr = PROCEDURE(control: ControlHandle;
    info: ControlBackgroundPtr);
```

Universal Procedure Pointer Types For User Pane Functions

```
ControlUserPaneDrawUPP = UniversalProcPtr;
ControlUserPaneHitTestUPP = UniversalProcPtr;
ControlUserPaneTrackingUPP = UniversalProcPtr;
ControlUserPaneIdleUPP = UniversalProcPtr;
ControlUserPaneKeyDownUPP = UniversalProcPtr;
ControlUserPaneActivateUPP = UniversalProcPtr;
ControlUserPaneFocusUPP = UniversalProcPtr;
ControlUserPaneBackgroundUPP = UniversalProcPtr;
```

Functions

Give Idle Time To Controls

```
PROCEDURE IdleControls(inWindow: WindowPtr);
```

Send Keyboard Event to Control With keyboard Focus

```
FUNCTION  HandleControlKey(inControl: ControlHandle; inKeyCode: SInt16; inCharCode: SInt16;
               inModifiers: SInt16): SInt16;
```

Set the Background For a Control

```
FUNCTION  SetUpControlBackground(inControl: ControlHandle; inDepth: SInt16;
               inIsColorDevice: BOOLEAN): OSErr;
```

KeyBoard Focus

```
FUNCTION  GetKeyboardFocus(inWindow: WindowPtr; VAR outControl: ControlHandle): OSErr;
FUNCTION  SetKeyboardFocus(inWindow: WindowPtr; inControl: ControlHandle;
               inPart: ControlFocusPart): OSErr;
FUNCTION  AdvanceKeyboardFocus(inWindow: WindowPtr): OSErr;
FUNCTION  ReverseKeyboardFocus(inWindow: WindowPtr): OSErr;
FUNCTION  ClearKeyboardFocus(inWindow: WindowPtr): OSErr;
```

Control Features

```
FUNCTION  GetControlFeatures(inControl: ControlHandle; VAR outFeatures: UInt32): OSErr;
```

Validating Controls

```
Boolean    IsValidControlhandle(Controlhandle theControl);
```

Routine Descriptors — User Pane Functions

```
FUNCTION  NewControlUserPaneDrawProc(userRoutine: ControlUserPaneDrawProcPtr):
               ControlUserPaneDrawUPP;
FUNCTION  NewControlUserPaneHitTestProc(userRoutine: ControlUserPaneHitTestProcPtr):
               ControlUserPaneHitTestUPP;
FUNCTION  NewControlUserPaneTrackingProc(userRoutine: ControlUserPaneTrackingProcPtr):
               ControlUserPaneTrackingUPP;
FUNCTION  NewControlUserPaneIdleProc(userRoutine: ControlUserPaneIdleProcPtr):
               ControlUserPaneIdleUPP;
FUNCTION  NewControlUserPaneKeyDownProc(userRoutine: ControlUserPaneKeyDownProcPtr):
               ControlUserPaneKeyDownUPP;
FUNCTION  NewControlUserPaneActivateProc(userRoutine: ControlUserPaneActivateProcPtr):
               ControlUserPaneActivateUPP;
FUNCTION  NewControlUserPaneFocusProc(userRoutine: ControlUserPaneFocusProcPtr):
               ControlUserPaneFocusUPP;
FUNCTION  NewControlUserPaneBackgroundProc(userRoutine: ControlUserPaneBackgroundProcPtr):
               ControlUserPaneBackgroundUPP;
```

Demonstration Program

```
{ ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
// Controls3Program.p
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
//
// This program demonstrates the creation and handling of those controls not demonstrated
// in the programs Controls1 and Controls2 (Chapter 7), with the exception of List boxes
// and determinate progress bars.
//
// The program utilises the following resources:
//
// •   An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit, and Demonstration
//     menus (preload, non-purgeable).
//
// •   'MENU' resources (non-purgeable) for bevel button menus and for a pop-up group box.
//
// •   A 'WIND' resource (purgeable) (initially not visible).
//
// •   'DLOG' resources and associated 'DITL', 'dlgx' and 'dftb' rersources (purgeable).
//
```

```
// •   'CNTL' resources (purgeable).
//
// •   A 'tab#' resource (purgeable.
//
// •   An icon family resource (purgeable).
//
// •   'PICT' resources (purgeable).
//
// •   'cicn' resources (purgeable).
//
// •   'STR#' resources (purgeable).
//
// •   'hrct' and an 'hwin' resources (preload, purgeable), which provide help
//     balloons describing  the various controls.
//
// •   A 'SIZE' resource with the acceptSuspendResumeEvents, doesActivateOnFGSwitch,
//     and is32BitCompatible flags set.
//
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ }

program Controls3Program;

uses

    { Universal Interfaces. }
    MacTypes, Events,

    { Other project files. }
    Controls3;

var

    mainMenubarHdl : Handle;
    mainMenuHdl : MenuHandle;
    mainEvent : EventRecord;
    mainErr : OSErr;

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ main program block

begin

    gBevelAndImageActive := false;
    gGroupArrowsProgressActive := false;
    gSlidersActive := false;
    gDrawActivated := true;

    gRedColour.red := $0000;
    gRedColour.green := $0000;
    gRedColour.blue := $0000;

    gBlueColour.red := $0000;
    gBlueColour.green := $0000;
    gBlueColour.blue := $0000;

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

    DoInitManagers;

    // ...... cause the Appearance-compliant menu bar definition function to be called directly

    mainErr := RegisterAppearanceClient;

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

    mainMenubarHdl := GetNewMBar(rMenubar);
    if (mainMenubarHdl = nil) then
        begin
        ExitToShell;
        end;
    SetMenuBar(mainMenubarHdl);
    DrawMenuBar;

    mainMenuHdl := GetMenuHandle(mApple);
    if (mainMenuHdl = nil) then
        begin
        ExitToShell;
        end
```

```
    else begin
        AppendResMenu(mainMenuHdl,'DRVR');
        end;

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

    gDone := false;

    while not gDone do
        begin
        if WaitNextEvent(everyEvent, mainEvent, MAXLONG, nil) then
            begin
            DoEvents(mainEvent);
            end;
        end;
end.
    { of main program block }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

{ ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
  Controls3.p
  ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ }

unit Controls3;


interface

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

uses

    { Universal Interfaces. }
    Appearance, Devices, Dialogs, Events, Fonts, LowMem, MacTypes, MacWindows, Menus,
    NumberFormatting, Processes, Resources, Sound, TextUtils, ToolUtils;

// ..............................................................................................................................................................
......................................... constants

const

rMenubar = 128;
mApple = 128;
iAbout = 1;
mFile = 129;
iQuit = 11;
mDemonstration = 131;
iBevelAndImage = 1;
iTabEditClock = 2;
iGroupArrowsProgress = 3;
iSliders = 4;
rBevelImageWindow = 128;
cBevelButton1 = 128;
cBevelButton2 = 129;
cBevelButton3 = 130;
cBevelButton4 = 131;
cBevelButton5 = 132;
cBevelButton6 = 133;
cBevelButton7 = 134;
cBevelButton8 = 135;
cBevelButton9 = 136;
cBevelButton10 = 137;
cBevelButton11 = 138;
cBevelButton12 = 139;
cBevelButton13 = 140;
cBevelButton14 = 141;
cBevelButton15 = 142;
cBevelButton16 = 143;
cBevelButton17 = 144;
cBevelButton18 = 145;
cBevelButton19 = 146;
cBevelButton20 = 147;
cBevelButton21 = 148;
cImageWell1 = 149;
```

```
cImageWell2 = 150;
cPicture1 = 151;
cPicture2 = 152;
cColourIcon1 = 153;
cColourIcon2 = 154;
cIconSuite1 = 155;
cIconSuite2 = 156;
cWindowHeader = 157;
rPartCodeStrings = 128;
rGraphicAlignStrings = 129;
rTextAlignStrings = 130;
rTextPlacementStrings = 131;
rTabEditClockDialog = 128;
iTabs = 2;
tabEditText = 1;
tabClocks = 2;
iEditTextUserPane = 3;
iEditText1 = 5;
iEditText2 = 7;
iEditText3 = 9;
iExtractEditText = 10;
iClocksUserPane = 12;
iImageWellEditText = 11;
iClocks1 = 14;
iClocks2 = 16;
iClocks3 = 18;
iExtractClocks = 19;
iImageWellClocks = 20;

kLeftArrow = $1C;
kRightArrow = $1D;
kUpArrow = $1E;
kDownArrow = $1F;
kBackspace = $08;

rGroupArrowsProgDialog = 129;
iCheckboxGroup = 2;
iRadioGroupColour = 3;
iStaticTextColourDepth = 7;
iPopupGroup = 8;
iUserPaneNamesInitials = 9;
iRadioGroupNames = 10;
iCheckboxShowInitials = 13;
iUserPaneScoreAverage = 15;
iRadioGroupScores = 16;
iCheckboxShowAverages = 19;
iStaticTextCache = 26;
iLittleArrows = 27;
iPushButtonExtract = 28;
iImageWell = 29;
iDisclosureTriangle = 31;
iStaticTextDisclosure = 32;
iProgressBar = 34;
rSlidersDialog = 131;
iSlider1 = 2;
iSlider2 = 3;
iSlider3 = 4;
iSlider4 = 5;
iSlider1StaticText = 9;
iSlider2StaticText = 11;
iSlider3StaticText = 13;
iSlider4StaticText = 15;
iSlider5 = 17;
iUserPane1 = 18;
iSlider6 = 19;
MAXLONG = $7FFFFFFF;


type

//
.....
..... type definitions

BevelDocRecord = record
    bevelButton1Hdl : ControlHandle;
    bevelButton2Hdl : ControlHandle;
    bevelButton3Hdl : ControlHandle;
    bevelButton4Hdl : ControlHandle;
    bevelButton5Hdl : ControlHandle;
```

```
    bevelButton6Hdl : ControlHandle;
    bevelButton7Hdl : ControlHandle;
    bevelButton8Hdl : ControlHandle;
    bevelButton9Hdl : ControlHandle;
    bevelButton10Hdl : ControlHandle;
    bevelButton11Hdl : ControlHandle;
    bevelButton12Hdl : ControlHandle;
    bevelButton13Hdl : ControlHandle;
    bevelButton14Hdl : ControlHandle;
    bevelButton15Hdl : ControlHandle;
    bevelButton16Hdl : ControlHandle;
    bevelButton17Hdl : ControlHandle;
    bevelButton18Hdl : ControlHandle;
    bevelButton19Hdl : ControlHandle;
    bevelButton20Hdl : ControlHandle;
    bevelButton21Hdl : ControlHandle;
    imageWell1Hdl : ControlHandle;
    imageWell2Hdl : ControlHandle;
    picture1Hdl : ControlHandle;
    picture2Hdl : ControlHandle;
    colourIcon1Hdl : ControlHandle;
    colourIcon2Hdl : ControlHandle;
    iconSuite1Hdl : ControlHandle;
    iconSuite2Hdl : ControlHandle;
    windowHeaderHdl : ControlHandle;
    end;

BevelDocPointer = ^BevelDocRecord;
BevelDocHandle = ^BevelDocPointer;

// ......................................................................................................................................................................
............... exported variables

var

gInBackground : boolean;
gDone : boolean;
gDrawActivated : boolean;
gCurrentString : Str255;
gBevelAndImageActive : boolean;
gGroupArrowsProgressActive : boolean;
gSlidersActive : boolean;
gPixelDepth : integer;
gIsColourDevice : boolean;
gRedColour : RGBColor;
gBlueColour : RGBColor;

// ......................................................................................................................................................................
.................. exported routines

procedure DoInitManagers;
procedure DoEvents({const} var theEvent : EventRecord);
procedure DoMouseDown({const} var theEvent : EventRecord);
procedure DoMenuChoice(menuChoice : SInt32);
procedure DoUpdate({const} var theEvent : EventRecord);
procedure DoActivate({const} var theEvent : EventRecord);
procedure DoActivateWindow(theWindowPtr : WindowPtr; becomingActive : boolean);
procedure DoGetDepthAndDevice;


implementation

uses

    { Other project files. }
    BevelImagePictIcon, GroupArrowsProgress, Sliders, TabEditClock;

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoInitManagers

procedure DoInitManagers;
    begin
    MaxApplZone;
    MoreMasters;

    InitGraf(@qd.thePort);
    InitFonts;
    InitWindows;
    InitMenus;
```

```
    TEInit;
    InitDialogs(nil);

    InitCursor;
    FlushEvents(everyEvent, 0);
    end;
        { of procedure DoInitManagers }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoEvents

procedure DoEvents({const}var theEvent : EventRecord);
    var
    charCode : SInt8;
    menuChoice : SInt32;
    menuID, menuItem : SInt16;

    begin
    case (theEvent.what) of

        keyDown, autoKey: begin
            charCode := SInt8(BAnd(theEvent.message, charCodeMask));
            if (BAnd(theEvent.modifiers, cmdKey) <> 0) then
                begin
                menuChoice := MenuEvent(theEvent);
                menuID := HiWord(menuChoice);
                menuItem := LoWord(menuChoice);
                if ((menuID = mFile) and (menuItem  = iQuit)) then
                    begin
                    gDone := true;
                    end;
                end;
            end;

        mouseDown: begin
            DoMouseDown(theEvent);
            end;

        updateEvt: begin
            DoUpdate(theEvent);
            end;

        activateEvt: begin
            DoActivate(theEvent);
            end;

        osEvt: begin
            if (BAnd(BSR(theEvent.message, 24), $000000FF) = suspendResumeMessage) then
                begin
                gInBackground := BAnd(theEvent.message, resumeFlag) = 0;
                DoActivateWindow(FrontWindow, not gInBackground);
                end;

            HiliteMenu(0);
            end;

        otherwise begin
            end;
        end;
    end;
        { of procedure DoEvents }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoMouseDown

procedure DoMouseDown({const}var theEvent : EventRecord);
    var
    partCode : SInt16;
    theWindowPtr : WindowPtr;
    menuHdl : MenuHandle;

    begin
    partCode := FindWindow(theEvent.where, theWindowPtr);

    case partCode of

        inMenuBar: begin
            menuHdl := GetMenuHandle(mDemonstration);
            if gBevelAndImageActive then
                begin
                DisableItem(menuHdl, iBevelAndImage);
                end
```

```
      else begin
        EnableItem(menuHdl, iBevelAndImage);
        end;

      DoMenuChoice(MenuSelect(theEvent.where));
      end;

    inContent: begin
      if (theWindowPtr <> FrontWindow) then
        begin
        SelectWindow(theWindowPtr);
        end
      else begin
        if (gBevelAndImageActive) then
          begin
          DoBevelImagePictIconContent(theEvent, theWindowPtr);
          end;
        end;
      end;

    inDrag: begin
      DragWindow(theWindowPtr, theEvent.where, qd.screenBits.bounds);
      end;

    inGoAway: begin
      if TrackGoAway(theWindowPtr, theEvent.where) then
        begin
        DisposeWindow(theWindowPtr);
        gBevelAndImageActive := false;
        end;
      end;

    otherwise begin
      end;

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

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoMenuChoice

procedure DoMenuChoice(menuChoice : SInt32);
  var
  menuID, menuItem : SInt16;
  itemName : Str255;
  daDriverRefNum : SInt16;

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

  if (menuID = 0) then
    begin
    Exit(DoMenuChoice);
    end;

  case menuID of

    mApple: begin
      if (menuItem = iAbout) then
        begin
        SysBeep(10);
        end
      else begin
        GetMenuItemText(GetMenuHandle(mApple), menuItem, itemName);
        daDriverRefNum := OpenDeskAcc(itemName);
        end;
      end;

    mFile: begin
      if (menuItem = iQuit) then
        begin
        gDone := true;
        end;
      end;

    mDemonstration: begin
      case menuItem of
```

```
            iBevelAndImage: begin
                gBevelAndImageActive := true;
                DoBevelImagePictIcon;
                end;

            iTabEditClock: begin
                DoTabEditClock;
                end;

            iGroupArrowsProgress: begin
                gGroupArrowsProgressActive := true;
                DoGroupArrowsProgress;
                end;

            iSliders: begin
                gSlidersActive := true;
                DoSliderUserPane;
                end;

            otherwise begin
                end;
            end;
                { of case statement }
        end;

    otherwise begin
        end;

    end;
        { of case statement }

  HiliteMenu(0);
  end;
    { of procedure DoMenuChoice }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoUpdate

procedure DoUpdate({const}var theEvent : EventRecord);
  var
  theWindowPtr : WindowPtr;
  drawMode : boolean;

  begin
  drawMode := false;

  theWindowPtr := WindowPtr(theEvent.message);

  BeginUpdate(theWindowPtr);

  if gBevelAndImageActive then
    begin
    SetPort(theWindowPtr);
    UpdateControls(theWindowPtr, theWindowPtr^.visRgn);
    if (theWindowPtr = FrontWindow) then
      begin
      drawMode := true;
      end;
    DoDrawMessage(theWindowPtr, drawMode);
    DoDrawLegends(drawMode);
    end;

  EndUpdate(theWindowPtr);
  end;
    { of procedure DoUpdate }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoActivate

procedure DoActivate({const}var theEvent : EventRecord);
  var
  theWindowPtr : WindowPtr;
  becomingActive : Boolean;

  begin
  theWindowPtr := WindowPtr(theEvent.message);
  becomingActive := (BAnd(theEvent.modifiers, activeFlag) = activeFlag);
  DoActivateWindow(theWindowPtr, becomingActive);
  end;
    { of procedure DoActivate }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoActivateWindow
```

```
procedure DoActivateWindow(theWindowPtr : WindowPtr; becomingActive : boolean);
    var
    controlHdl : ControlHandle;
    ignoredErr : OSErr;

    begin
    ignoredErr := GetRootControl(theWindowPtr, controlHdl);

    if becomingActive then
        begin
        if gBevelAndImageActive then
            begin
            ignoredErr := ActivateControl(controlHdl);
            DoDrawMessage(theWindowPtr, becomingActive);
            DoDrawLegends(becomingActive);
            end;
        end
    else begin
        if gBevelAndImageActive then
            begin
            ignoredErr := DeactivateControl(controlHdl);
            DoDrawMessage(theWindowPtr, becomingActive);
            DoDrawLegends(becomingActive);
            end;
        end;
    end;
        { of procedure DoActivateWindow }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoGetDepthAndDevice

procedure DoGetDepthAndDevice;
    var
    deviceHdl : GDHandle;

    begin
    deviceHdl := LMGetMainDevice;
    gPixelDepth := deviceHdl^^.gdPMap^^.pixelSize;
    if BitTst(@deviceHdl^^.gdFlags, gdDevType) then
        begin
        gIsColourDevice := true;
        end;
    end;
        { of procedure DoGetDepthAndDevice }

end.
    { of unit Controls3 }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

{ ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
  BevelImagePictIcon.p
  ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ }

unit BevelImagePictIcon;


interface

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

uses

    { Other project files. }
    Controls3;

procedure DoBevelImagePictIcon;
procedure DoGetControls(theWindowPtr : WindowPtr);
procedure DoBevelImagePictIconContent({const} var theEvent : EventRecord;
                        theWindowPtr : WindowPtr);
procedure DoDrawPartCode(theWindowPtr : WindowPtr; windowHeaderHdl : ControlHandle;
                        partCode, menuItem : SInt16);
procedure DoGraphicAlignment(theWindowPtr : WindowPtr;
                        controlHdl, windowHeaderHdl : ControlHandle);
procedure DoTextAlignment(theWindowPtr : WindowPtr;
                        controlHdl, windowHeaderHdl : ControlHandle);
procedure DoTextOffset(theWindowPtr : WindowPtr; controlHdl, windowHeaderHdl : ControlHandle);
procedure DoTextPlacement(theWindowPtr : WindowPtr;
```

```
                        controlHdl, windowHeaderHdl : ControlHandle);
procedure DoDrawMessage(theWindowPtr : WindowPtr; inState : boolean);
procedure DoDrawLegends(inState : boolean);


implementation

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoBevelImagePictIcon

procedure DoBevelImagePictIcon;
   var
   theWindowPtr : WindowPtr;
   bevelDocRecordHdl : BevelDocHandle;
   ignoredErr : OSErr;

   begin
   // ...................................................................................................................... initial advisory text for window
header

   gCurrentString := 'Balloon help is available';

   // ......... open a window, set font size, set Appearance-compliant colour/pattern for window

   theWindowPtr := GetNewCWindow(rBevelImageWindow, nil, WindowPtr(-1));
   if (theWindowPtr = nil) then
      begin
      ExitToShell;
      end;

   SetPort(theWindowPtr);
   TextSize(10);

   ignoredErr := SetThemeWindowBackground(theWindowPtr, kThemeActiveDialogBackgroundBrush,
                                                            true);

   // .................. get block for document structure, assign handle to window record refCon field

   bevelDocRecordHdl := BevelDocHandle(NewHandle(sizeof(BevelDocRecord)));
   if (bevelDocRecordHdl = nil) then
      begin
      ExitToShell;
      end;

   SetWRefCon(theWindowPtr, SInt32(bevelDocRecordHdl));

   // ......................................................................................... get controls, adjust scroll bars, and show window

   DoGetControls(theWindowPtr);
   ShowWindow(theWindowPtr);

   // .......................... get pixel depth and whether colour device for certain Appearance functions

   DoGetDepthAndDevice;
   end;
      { of procedure DoBevelImagePictIcon }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoGetControls

procedure DoGetControls(theWindowPtr : WindowPtr);
   var
   controlHdl : ControlHandle;
   bevelDocHdl : BevelDocHandle;
   textPlacement : ControlButtonTextPlacement;
   centrePopupGlyph : boolean;
   controlFontStyleStruc : ControlFontStyleRec;
   ignoredErr : OSErr;

   begin
   textPlacement := kControlBevelButtonPlaceAboveGraphic;
   centrePopupGlyph := true;

   // ............................. create root control for window, get handle to windows document structure

   ignoredErr := CreateRootControl(theWindowPtr, controlHdl);

   bevelDocHdl := BevelDocHandle(GetWRefCon(theWindowPtr));

   //
..................................................................................................................................................................
.............. get the controls
```

```
bevelDocHdl^^.bevelButton1Hdl := GetNewControl(cBevelButton1, theWindowPtr);
if (bevelDocHdl^^.bevelButton1Hdl = nil) then
    begin
    ExitToShell;
    end;

bevelDocHdl^^.bevelButton2Hdl := GetNewControl(cBevelButton2, theWindowPtr);
if (bevelDocHdl^^.bevelButton2Hdl = nil) then
    begin
    ExitToShell;
    end;

bevelDocHdl^^.bevelButton3Hdl := GetNewControl(cBevelButton3, theWindowPtr);
if (bevelDocHdl^^.bevelButton3Hdl = nil) then
    begin
    ExitToShell;
    end;

bevelDocHdl^^.bevelButton4Hdl := GetNewControl(cBevelButton4, theWindowPtr);
if (bevelDocHdl^^.bevelButton4Hdl = nil) then
    begin
    ExitToShell;
    end;

bevelDocHdl^^.bevelButton5Hdl := GetNewControl(cBevelButton5, theWindowPtr);
if (bevelDocHdl^^.bevelButton5Hdl = nil) then
    begin
    ExitToShell;
    end;

bevelDocHdl^^.bevelButton6Hdl := GetNewControl(cBevelButton6, theWindowPtr);
if (bevelDocHdl^^.bevelButton6Hdl = nil) then
    begin
    ExitToShell;
    end;

bevelDocHdl^^.bevelButton7Hdl := GetNewControl(cBevelButton7, theWindowPtr);
if (bevelDocHdl^^.bevelButton7Hdl = nil) then
    begin
    ExitToShell;
    end;

bevelDocHdl^^.bevelButton8Hdl := GetNewControl(cBevelButton8, theWindowPtr);
if (bevelDocHdl^^.bevelButton8Hdl = nil) then
    begin
    ExitToShell;
    end;

bevelDocHdl^^.bevelButton9Hdl := GetNewControl(cBevelButton9, theWindowPtr);
if (bevelDocHdl^^.bevelButton9Hdl = nil) then
    begin
    ExitToShell;
    end;

bevelDocHdl^^.bevelButton10Hdl := GetNewControl(cBevelButton10, theWindowPtr);
if (bevelDocHdl^^.bevelButton10Hdl = nil) then
    begin
    ExitToShell;
    end;

bevelDocHdl^^.bevelButton11Hdl := GetNewControl(cBevelButton11, theWindowPtr);
if (bevelDocHdl^^.bevelButton11Hdl = nil) then
    begin
    ExitToShell;
    end;

bevelDocHdl^^.bevelButton12Hdl := GetNewControl(cBevelButton12, theWindowPtr);
if (bevelDocHdl^^.bevelButton12Hdl = nil) then
    begin
    ExitToShell;
    end;

bevelDocHdl^^.bevelButton13Hdl := GetNewControl(cBevelButton13, theWindowPtr);
if (bevelDocHdl^^.bevelButton13Hdl = nil) then
    begin
    ExitToShell;
    end;

bevelDocHdl^^.bevelButton14Hdl := GetNewControl(cBevelButton14, theWindowPtr);
```

```
    if (bevelDocHdl^^.bevelButton14Hdl = nil) then
        begin
        ExitToShell;
        end;

    bevelDocHdl^^.bevelButton15Hdl := GetNewControl(cBevelButton15, theWindowPtr);
    if (bevelDocHdl^^.bevelButton15Hdl = nil) then
        begin
        ExitToShell;
        end;

    bevelDocHdl^^.bevelButton16Hdl := GetNewControl(cBevelButton16, theWindowPtr);
    if (bevelDocHdl^^.bevelButton16Hdl = nil) then
        begin
        ExitToShell;
        end;

    bevelDocHdl^^.bevelButton17Hdl := GetNewControl(cBevelButton17, theWindowPtr);
    if (bevelDocHdl^^.bevelButton17Hdl = nil) then
        begin
        ExitToShell;
        end;

    bevelDocHdl^^.bevelButton18Hdl := GetNewControl(cBevelButton18, theWindowPtr);
    if (bevelDocHdl^^.bevelButton18Hdl = nil) then
        begin
        ExitToShell;
        end;

    bevelDocHdl^^.bevelButton19Hdl := GetNewControl(cBevelButton19, theWindowPtr);
    if (bevelDocHdl^^.bevelButton19Hdl = nil) then
        begin
        ExitToShell;
        end;

    bevelDocHdl^^.bevelButton20Hdl := GetNewControl(cBevelButton20, theWindowPtr);
    if (bevelDocHdl^^.bevelButton20Hdl = nil) then
        begin
        ExitToShell;
        end;

    bevelDocHdl^^.bevelButton21Hdl := GetNewControl(cBevelButton21, theWindowPtr);
    if (bevelDocHdl^^.bevelButton21Hdl = nil) then
        begin
        ExitToShell;
        end;

    bevelDocHdl^^.imageWell1Hdl := GetNewControl(cImageWell1, theWindowPtr);
    if (bevelDocHdl^^.imageWell1Hdl = nil) then
        begin
        ExitToShell;
        end;

    bevelDocHdl^^.imageWell2Hdl := GetNewControl(cImageWell2, theWindowPtr);
    if (bevelDocHdl^^.imageWell2Hdl = nil) then
        begin
        ExitToShell;
        end;

    bevelDocHdl^^.picture1Hdl := GetNewControl(cPicture1, theWindowPtr);
    if (bevelDocHdl^^.picture1Hdl = nil) then
        begin
        ExitToShell;
        end;

    bevelDocHdl^^.picture2Hdl := GetNewControl(cPicture2, theWindowPtr);
    if (bevelDocHdl^^.picture2Hdl = nil) then
        begin
        ExitToShell;
        end;

    bevelDocHdl^^.colourIcon1Hdl := GetNewControl(cColourIcon1, theWindowPtr);
    if (bevelDocHdl^^.colourIcon1Hdl = nil) then
        begin
        ExitToShell;
        end;

    bevelDocHdl^^.colourIcon2Hdl := GetNewControl(cColourIcon2, theWindowPtr);
    if (bevelDocHdl^^.colourIcon2Hdl = nil) then
        begin
```

```
    ExitToShell;
    end;

bevelDocHdl^^.iconSuite1Hdl := GetNewControl(cIconSuite1, theWindowPtr);
if (bevelDocHdl^^.iconSuite1Hdl = nil) then
    begin
    ExitToShell;
    end;

bevelDocHdl^^.iconSuite2Hdl := GetNewControl(cIconSuite2, theWindowPtr);
if (bevelDocHdl^^.iconSuite2Hdl = nil) then
    begin
    ExitToShell;
    end;

bevelDocHdl^^.windowHeaderHdl := GetNewControl(cWindowHeader, theWindowPtr);
if (bevelDocHdl^^.windowHeaderHdl = nil) then
    begin
    ExitToShell;
    end;

// ..... set text placement for 2nd and 21st bevel button

ignoredErr := SetControlData(bevelDocHdl^^.bevelButton2Hdl, kControlNoPart,
                        kControlBevelButtonTextPlaceTag, sizeof(textPlacement),
                        Ptr(@textPlacement));
ignoredErr := SetControlData(bevelDocHdl^^.bevelButton21Hdl, kControlNoPart,
                        kControlBevelButtonTextPlaceTag, sizeof(textPlacement),
                        Ptr(@textPlacement));

// ..... set position of pop-up arrow in 6th bevel button

ignoredErr := SetControlData(bevelDocHdl^^.bevelButton6Hdl, kControlNoPart,
                        kControlBevelButtonCenterPopupGlyphTag, sizeof(centrePopupGlyph),
                        Ptr(@centrePopupGlyph));

// ..... set font for 20th bevel button to small bold system font

controlFontStyleStruc.flags := kControlUseFontMask;
controlFontStyleStruc.font := kControlFontSmallBoldSystemFont;
    ignoredErr := SetControlFontStyle(bevelDocHdl^^.bevelButton20Hdl, controlFontStyleStruc);

    // ..... set 3rd bevel button to the
mixed state

    SetControlValue(bevelDocHdl^^.bevelButton3Hdl, 2);
    end;
        { of procedure DoGetControls }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoBevelImagePictIconContent

procedure DoBevelImagePictIconContent({const} var theEvent : EventRecord;
                            theWindowPtr : WindowPtr);
    var
    bevelDocHdl : BevelDocHandle;
    controlHdl : ControlHandle;
    ignoredErr : OSErr;
    partCode, menuItem : SInt16;
    actualSize : Size;

    begin
    bevelDocHdl := BevelDocHandle((GetWRefCon(theWindowPtr)));

    GlobalToLocal(theEvent.where);
    partCode := FindControl(theEvent.where, theWindowPtr, controlHdl);

    DoDrawPartCode(theWindowPtr, bevelDocHdl^^.windowHeaderHdl, partCode, 1234);

    if (partCode <> kControlNoPart ) then
        begin
        partCode := TrackControl(controlHdl, theEvent.where, nil);

        if ((controlHdl = bevelDocHdl^^.bevelButton1Hdl) or
            (controlHdl = bevelDocHdl^^.bevelButton2Hdl) or
            (controlHdl = bevelDocHdl^^.bevelButton3Hdl) or
            (controlHdl = bevelDocHdl^^.bevelButton4Hdl) or
            (controlHdl = bevelDocHdl^^.bevelButton9Hdl) or
            (controlHdl = bevelDocHdl^^.bevelButton13Hdl) or
            (controlHdl = bevelDocHdl^^.bevelButton14Hdl) or
            (controlHdl = bevelDocHdl^^.bevelButton15Hdl) or
```

```
            (controlHdl = bevelDocHdl^^.bevelButton16Hdl) or
            (controlHdl = bevelDocHdl^^.bevelButton17Hdl)) then
            begin
            DoDrawPartCode(theWindowPtr, bevelDocHdl^^.windowHeaderHdl, partCode, 4321);
            end
        else if ((controlHdl = bevelDocHdl^^.bevelButton5Hdl) or
                (controlHdl = bevelDocHdl^^.bevelButton6Hdl) or
                (controlHdl = bevelDocHdl^^.bevelButton7Hdl) or
                (controlHdl = bevelDocHdl^^.bevelButton8Hdl)) then
            begin
            if (partCode = kControlMenuPart) then
                begin
                ignoredErr := GetControlData(controlHdl, kControlMenuPart,
                    kControlBevelButtonMenuValueTag, sizeof(menuItem), Ptr(@menuItem), actualSize);
                DoDrawPartCode(theWindowPtr, bevelDocHdl^^.windowHeaderHdl, partCode, menuItem);
                end
            else begin
                DoDrawPartCode(theWindowPtr, bevelDocHdl^^.windowHeaderHdl, partCode, 4321);
                end;
            end
        else if ((controlHdl = bevelDocHdl^^.bevelButton10Hdl) or
                (controlHdl = bevelDocHdl^^.bevelButton11Hdl) or
                (controlHdl = bevelDocHdl^^.bevelButton12Hdl)) then
            begin
            if (partCode <> kControlNoPart) then
                begin
                SetControlValue(bevelDocHdl^^.bevelButton10Hdl, 0);
                SetControlValue(bevelDocHdl^^.bevelButton11Hdl, 0);
                SetControlValue(bevelDocHdl^^.bevelButton12Hdl, 0);
                SetControlValue(controlHdl, 1);
                end;
            DoDrawPartCode(theWindowPtr, bevelDocHdl^^.windowHeaderHdl, partCode, 4321);
            end
        else if (controlHdl = bevelDocHdl^^.bevelButton18Hdl) then
            begin
            DoDrawPartCode(theWindowPtr, bevelDocHdl^^.windowHeaderHdl, partCode, 4321);
            DoGraphicAlignment(theWindowPtr, controlHdl, bevelDocHdl^^.windowHeaderHdl);
            end
        else if (controlHdl = bevelDocHdl^^.bevelButton19Hdl) then
            begin
            DoDrawPartCode(theWindowPtr, bevelDocHdl^^.windowHeaderHdl, partCode, 4321);
            DoTextAlignment(theWindowPtr, controlHdl, bevelDocHdl^^.windowHeaderHdl);
            end
        else if (controlHdl = bevelDocHdl^^.bevelButton20Hdl) then
            begin
            DoDrawPartCode(theWindowPtr, bevelDocHdl^^.windowHeaderHdl, partCode, 4321);
            DoTextOffset(theWindowPtr, controlHdl, bevelDocHdl^^.windowHeaderHdl);
            end
        else if (controlHdl = bevelDocHdl^^.bevelButton21Hdl) then
            begin
            DoDrawPartCode(theWindowPtr, bevelDocHdl^^.windowHeaderHdl, partCode, 4321);
            DoTextPlacement(theWindowPtr, controlHdl, bevelDocHdl^^.windowHeaderHdl);
            end
        else begin
            DoDrawPartCode(theWindowPtr, bevelDocHdl^^.windowHeaderHdl, partCode, 4321);
            end;
        end;
    end;
        { of procedure DoBevelImagePictIconContent }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoDrawPartCode

procedure DoDrawPartCode(theWindowPtr : WindowPtr; windowHeaderHdl : ControlHandle;
                        partCode, menuItem : SInt16);
    var
    stringIndex : SInt16;
    theString : Str255;

    begin
    if (partCode = kControlNoPart) then
        begin
        stringIndex := 1;
        end
    else if (partCode = kControlMenuPart) then
        begin
        stringIndex := 2;
        end
    else if (partCode = kControlTrianglePart) then
        begin
        stringIndex := 3;
```

```
        end
    else if (partCode = kControlEditTextPart) then
        begin
        stringIndex := 4;
        end
    else if (partCode = kControlPicturePart) then
        begin
        stringIndex := 5;
        end
    else if (partCode = kControlIconPart) then
        begin
        stringIndex := 6;
        end
    else if (partCode = kControlClockPart) then
        begin
        stringIndex := 7;
        end
    else if (partCode = kControlListBoxPart) then
        begin
        stringIndex := 8;
        end
    else if (partCode = kControlListBoxDoubleClickPart) then
        begin
        stringIndex := 9;
        end
    else if (partCode = kControlImageWellPart) then
        begin
        stringIndex := 10;
        end
    else if (partCode = kControlRadioGroupPart) then
        begin
        stringIndex := 11;
        end
    else if (partCode = kControlButtonPart) then
        begin
        stringIndex := 12;
        end
    else if (partCode = kControlIndicatorPart) then
        begin
        stringIndex := 13;
        end;

    if ((menuItem > 0) and (menuItem < 1234)) then
        begin
        gCurrentString := 'TrackControl returned ';
        GetIndString(theString, rPartCodeStrings, stringIndex);
        gCurrentString := gCurrentString + theString + '    GetControlData returned menu item ';
        NumToString(SInt32(menuItem), theString);
        gCurrentString := gCurrentString + theString;
        end
    else if ((menuItem = 1234) or (menuItem = 4321)) then
        begin
        if (menuItem = 1234) then
            begin
            gCurrentString := 'Mouse-down in ';
            end
        else if (menuItem = 4321) then
            begin
            gCurrentString := 'TrackControl returned ';
            end;
        GetIndString(theString, rPartCodeStrings, stringIndex);
        gCurrentString := gCurrentString + theString;
        end;

    Draw1Control(windowHeaderHdl);
    DoDrawMessage(theWindowPtr, not gInBackground);
    end;
        { of procedure DoDrawPartCode }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoGraphicAlignment

procedure DoGraphicAlignment(theWindowPtr : WindowPtr; controlHdl, windowHeaderHdl : ControlHandle);
    var
    a : SInt16;
    finalTicks : UInt32;
    alignmentConstant : ControlButtonGraphicAlignment;
    ignoredErr : OSErr;

    begin
    alignmentConstant := 0;
```

```
    SetCursor(GetCursor(watchCursor)^^);

    for a := 1 to 9 do
        begin
        Delay(60, finalTicks);
        alignmentConstant := alignmentConstant + 1;
        if (alignmentConstant = 9) then
            begin
            alignmentConstant := 0;
            end;
        ignoredErr := SetControlData(controlHdl, kControlNoPart,
                                kControlBevelButtonGraphicAlignTag,sizeof(alignmentConstant),
                                Ptr(@alignmentConstant));
        Draw1Control(controlHdl);
        Draw1Control(windowHeaderHdl);
        GetIndString(gCurrentString, rGraphicAlignStrings, a);
        DoDrawMessage(theWindowPtr, not gInBackground);
        end;

    SetCursor(qd.arrow);
    end;
        { of procedure DoGraphicAlignment }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoTextAlignment

procedure DoTextAlignment(theWindowPtr : WindowPtr; controlHdl,
                            windowHeaderHdl : ControlHandle);
    var
    a : SInt16;
    finalTicks : UInt32;
    alignmentConstant : ControlButtonTextAlignment;
    ignoredErr : OSErr;

    begin
    alignmentConstant := -3;

    SetCursor(GetCursor(watchCursor)^^);

    for a := 1 to 3 do
        begin
        Delay(60, finalTicks);
        alignmentConstant := alignmentConstant + 1;
        if (alignmentConstant = 0) then
            begin
            alignmentConstant := alignmentConstant + 1;
            end;
        ignoredErr := SetControlData(controlHdl, kControlNoPart, kControlBevelButtonTextAlignTag,
                            sizeof(alignmentConstant), Ptr(@alignmentConstant));
        Draw1Control(controlHdl);
        Draw1Control(windowHeaderHdl);
        GetIndString(gCurrentString, rTextAlignStrings, a);
        DoDrawMessage(theWindowPtr, not gInBackground);
        end;

    SetCursor(qd.arrow);
    end;
        { of procedure DoTextAlignment }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoTextOffset

procedure DoTextOffset(theWindowPtr : WindowPtr; controlHdl, windowHeaderHdl : ControlHandle);
    var
    alignmentConstant : ControlButtonTextAlignment;
    offset : SInt16;
    finalTicks : UInt32;
    ignoredErr : OSErr;

    begin
    SetCursor(GetCursor(watchCursor)^^);

    Draw1Control(windowHeaderHdl);
    gCurrentString := 'Offset from left';
    DoDrawMessage(theWindowPtr, not gInBackground);

    alignmentConstant := kControlBevelButtonAlignTextFlushLeft;
    ignoredErr := SetControlData(controlHdl, kControlNoPart, kControlBevelButtonTextAlignTag,
                            sizeof(alignmentConstant), Ptr(@alignmentConstant));
    Draw1Control(controlHdl);
```

```
    for offset := 1 to 27 do
      begin
      Delay(15, finalTicks);
      ignoredErr := SetControlData(controlHdl, kControlNoPart, kControlBevelButtonTextOffsetTag,
                             sizeof(offset), Ptr(@offset));
      Draw1Control(controlHdl);
      end;

    Delay(60, finalTicks);

    Draw1Control(windowHeaderHdl);
    gCurrentString := 'Offset from right';
    DoDrawMessage(theWindowPtr, not gInBackground);

    alignmentConstant := kControlBevelButtonAlignTextFlushRight;
    ignoredErr := SetControlData(controlHdl, kControlNoPart, kControlBevelButtonTextAlignTag,
                             sizeof(alignmentConstant), Ptr(@alignmentConstant));

    for offset :=0 to 13 do
      begin
      Delay(15, finalTicks);
      ignoredErr := SetControlData(controlHdl, kControlNoPart, kControlBevelButtonTextOffsetTag,
                             sizeof(offset), Ptr(@offset));
      Draw1Control(controlHdl);
      end;

    SetCursor(qd.arrow);
    end;
      { of procedure DoTextOffset }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoTextPlacement

procedure DoTextPlacement(theWindowPtr : WindowPtr; controlHdl, windowHeaderHdl : ControlHandle);
    var
    placementConstant : ControlButtonTextPlacement;
    finalTicks : UInt32;
    ignoredErr : OSErr;

    begin
    SetCursor(GetCursor(watchCursor)^^);

    for placementConstant := 1 to 4 do
      begin
      Delay(60, finalTicks);
      ignoredErr := SetControlData(controlHdl, kControlNoPart, kControlBevelButtonTextPlaceTag,
                             sizeof(placementConstant), Ptr(@placementConstant));
      Draw1Control(controlHdl);
      Draw1Control(windowHeaderHdl);
      GetIndString(gCurrentString, rTextPlacementStrings, placementConstant);
      DoDrawMessage(theWindowPtr, not gInBackground);
      end;

    SetCursor(qd.arrow);
    end;
      { of procedure DoTextPlacement }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoDrawMessage

procedure DoDrawMessage(theWindowPtr : WindowPtr; inState : boolean);
    var
    windowWidth, stringWidth : SInt16;
    ignoredStatus : OSStatus;

    begin
    if inState then
      begin
      ignoredStatus := SetThemeTextColor(kThemeActiveWindowHeaderTextColor, gPixelDepth, gIsColourDevice);
      end
    else begin
      ignoredStatus := SetThemeTextColor(kThemeInactiveWindowHeaderTextColor, gPixelDepth,
                                         gIsColourDevice);
      end;

    windowWidth := theWindowPtr^.portRect.right - theWindowPtr^.portRect.left;
    stringWidth := StringWidth(gCurrentString);
    MoveTo((windowWidth div 2) - (stringWidth div 2), 17);
    DrawString(gCurrentString);
    end;
      { of procedure DoDrawMessage }
```

```
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoDrawLegends

procedure DoDrawLegends(inState : boolean);
   var
   ignoredStatus : OSStatus;

   begin
   if inState then
      begin
      ignoredStatus := SetThemeTextColor(kThemeActiveWindowHeaderTextColor, gPixelDepth,
                                         gIsColourDevice);
      end
   else begin
      ignoredStatus := SetThemeTextColor(kThemeInactiveWindowHeaderTextColor, gPixelDepth,
                                         gIsColourDevice);
      end;

   MoveTo(30, 63);
   DrawString('Bevel sizes and bevel button content');
   MoveTo(313, 63);
   DrawString('Menu position and behaviour');
   MoveTo(30, 154);
   DrawString('Bevel button behaviour');
   MoveTo(313, 154);
   DrawString('Graphic/text alignment and offset');
   MoveTo(490, 154);
   DrawString('Text Placement');
   MoveTo(30, 245);
   DrawString('Image wells');
   MoveTo(168, 245);
   DrawString('Picture controls');
   MoveTo(313, 245);
   DrawString('Icon controls (cicn)');
   MoveTo(451, 245);
   DrawString('Icon controls (icon suite)');
   end;
      { of procedure DoDrawLegends }

end.
   { of unit BevelImagePictIcon }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

{ ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
  TabEditClock.p
  ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ }

unit TabEditClock;


interface

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

uses

   { Other project files. }
   Controls3;


procedure DoTabEditClock;
procedure DoExtractEditText(theDialogPtr : DialogPtr);
procedure DoExtractDateTime(theDialogPtr : DialogPtr);
function  NumericFilter(controlHdl : ControlHandle;
                  var keyCode, charCode, modifiers : SInt16) : ControlKeyFilterResult;
procedure EditTextValidator(controlHdl : ControlHandle);


implementation

uses

   { Universal Interfaces. }
   DateTimeUtils,

   { Other project files.}
   Callbacks;
```

```
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoTabEditClock

procedure DoTabEditClock;
  var
  theDialogPtr : DialogPtr;
  controlHdl : ControlHandle;
  initialName : Str255;
  selectionRec : ControlEditTextSelectionRec;
  numericFilterUPP : ControlKeyFilterUPP;
  eventFilterUPP : ModalFilterUPP;
  editTextValidatorUPP : ControlEditTextValidationUPP;
  tabHit, itemHit : SInt16;
  ignoredErr : OSErr;

  begin
  initialName := 'Your name here';

  if (FrontWindow <> nil) then
    begin
    DoActivateWindow(FrontWindow, false);
    end;

  theDialogPtr := GetNewDialog(rTabEditClockDialog, nil, WindowPtr(-1));
  if (theDialogPtr = nil) then
    begin
    ExitToShell;
    end;

  SetWTitle(theDialogPtr, 'Tabs, Edit Text Fields, and Clocks');

  // ........................................................................................................................ set default button and cursor
tracking

  ignoredErr := SetDialogDefaultItem(theDialogPtr, kStdOkItemIndex);
  ignoredErr := SetDialogTracksCursor(theDialogPtr, true);

  // ............................................................................................................................ hide user pane with
embedded clocks

  ignoredErr := GetDialogItemAsControl(theDialogPtr, iClocksUserPane, controlHdl);
  HideControl(controlHdl);

  // ................................... assign some text to the first edit text field and select the whole text

  ignoredErr := GetDialogItemAsControl(theDialogPtr, iEditText1, controlHdl);
  ignoredErr := SetControlData(controlHdl, kControlNoPart, kControlEditTextTextTag,
          SInt16(initialName[0]), @initialName[1]);
  selectionRec.selStart := 0;
  selectionRec.selEnd := 32767;
  ignoredErr := SetControlData(controlHdl, kControlNoPart, kControlEditTextSelectionTag,
              sizeof(selectionRec), @selectionRec);

  // ......................................................................... create routine descriptors for event filter and key filter

  numericFilterUPP := NewRoutineDescriptor(@NumericFilter, uppControlKeyFilterProcInfo, GetCurrentISA);
  eventFilterUPP := NewModalFilterProc(@EventFilter);
  editTextValidatorUPP := NewRoutineDescriptor(@EditTextValidator, uppControlEditTextValidationProcInfo,
GetCurrentISA);

  // ...................................... attach an edit text validation function to the first edit text control

  ignoredErr := GetDialogItemAsControl(theDialogPtr, iEditText1, controlHdl);
  ignoredErr := SetControlData(controlHdl, kControlNoPart, kControlEditTextValidationProcTag,
              sizeof(editTextValidatorUPP), @editTextValidatorUPP);

  // ......................................................................... attach a key filter function to the second edit text field

  ignoredErr := GetDialogItemAsControl(theDialogPtr, iEditText2, controlHdl);
  ignoredErr := SetControlData(controlHdl, kControlNoPart, kControlEditTextKeyFilterTag,
              sizeof(numericFilterUPP), @numericFilterUPP);

  // ..................................................................................................................... show dialog and enter modal
dialog loop

  ShowWindow(theDialogPtr);

  repeat
    begin
    ModalDialog(eventFilterUPP, itemHit);
```

```
    if (itemHit = iTabs) then
      begin
      ignoredErr := GetDialogItemAsControl(theDialogPtr, iTabs, controlHdl);
      tabHit := GetControlValue(controlHdl);

      if (tabHit = tabEditText) then
        begin
        SetWTitle(theDialogPtr, 'Tabs, Edit Text Fields, and Clocks');

        ignoredErr := GetDialogItemAsControl(theDialogPtr, iClocksUserPane, controlHdl);
        HideControl(controlHdl);

        ignoredErr := GetDialogItemAsControl(theDialogPtr, iEditTextUserPane, controlHdl);
        ignoredErr := ActivateControl(controlHdl);
        ShowControl(controlHdl);

        ignoredErr := GetDialogItemAsControl(theDialogPtr, iEditText1, controlHdl);
        ignoredErr := SetKeyboardFocus(theDialogPtr, controlHdl, kControlFocusNextPart);
        end
      else if (tabHit = tabClocks) then
        begin
        SetWTitle(theDialogPtr, 'Tabs, Clocks, and Edit Text Fields');

        ignoredErr := GetDialogItemAsControl(theDialogPtr, iEditTextUserPane, controlHdl);
        ignoredErr := DeactivateControl(controlHdl);
        HideControl(controlHdl);

        ignoredErr := GetDialogItemAsControl(theDialogPtr, iClocksUserPane, controlHdl);
        ShowControl(controlHdl);

        ignoredErr := GetDialogItemAsControl(theDialogPtr, iClocks1, controlHdl);
        ignoredErr := SetKeyboardFocus(theDialogPtr, controlHdl, kControlFocusNextPart);
        end;
      end
    else if (itemHit = iExtractEditText) then
      begin
      ignoredErr := GetDialogItemAsControl(theDialogPtr, iImageWellEditText, controlHdl);
      Draw1Control(controlHdl);
      DoExtractEditText(theDialogPtr);
      end
    else if (itemHit = iExtractClocks) then
      begin
      ignoredErr := GetDialogItemAsControl(theDialogPtr, iImageWellClocks, controlHdl);
      Draw1Control(controlHdl);
      DoExtractDateTime(theDialogPtr);
      end;
    end;
  until (itemHit = kStdOkItemIndex);

  DisposeDialog(theDialogPtr);
  DisposeRoutineDescriptor(numericFilterUPP);
  DisposeRoutineDescriptor(eventFilterUPP);
  end;
    { of procedure DoTabEditClock }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoExtractEditText

procedure DoExtractEditText(theDialogPtr : DialogPtr);
  var
  oldPort : GrafPtr;
  saveBackColour, whiteColour : RGBColor;
  iType : SInt16;
  theHandle : Handle;
  theRect : Rect;
  theString : Str255;
  controlHdl : ControlHandle;
  actualSize : Size;
  ignoredErr : OSErr;

  begin
  whiteColour.red := $FFFF;
  whiteColour.green := $FFFF;
  whiteColour.blue := $FFFF;

  GetPort(oldPort);
  SetPort(theDialogPtr);
  GetBackColor(saveBackColour);
  RGBBackColor(whiteColour);

  GetDialogItem(theDialogPtr, iEditText1, iType, theHandle, theRect);
```

```
    GetDialogItemText(theHandle, theString);
    MoveTo(100, 182);
    DrawString(theString);

    GetDialogItem(theDialogPtr, iEditText2, iType, theHandle, theRect);
    GetDialogItemText(theHandle, theString);
    MoveTo(100, 196);
    DrawString(theString);

    ignoredErr := GetDialogItemAsControl(theDialogPtr, iEditText3, controlHdl);
    ignoredErr := GetControlDataSize(controlHdl, kControlEditTextPart, kControlEditTextTextTag, actualSize);
    ignoredErr := GetControlData(controlHdl, kControlNoPart, kControlEditTextPasswordTag, actualSize,
                        Ptr(@theString[1]), actualSize);
    theString[0] := char(actualSize);
    MoveTo(100, 208);
    DrawString(theString);

    RGBBackColor(saveBackColour);
    SetPort(oldPort);
    end;
        { of procedure DoExtractEditText }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoExtractDateTime

procedure DoExtractDateTime(theDialogPtr : DialogPtr);
    var
    oldPort : GrafPtr;
    saveBackColour, whiteColour : RGBColor;
    controlHdl : ControlHandle;
    longDateTimeStruc : LongDateRec;
    second, minute, hour, day, month, year : SInt16;
    theString, tempString : Str255;
    actualSize : Size;
    ignoredErr : OSErr;

    begin
    whiteColour.red := $FFFF;
    whiteColour.green := $FFFF;
    whiteColour.blue := $FFFF;

    GetPort(oldPort);
    SetPort(theDialogPtr);
    GetBackColor(saveBackColour);
    RGBBackColor(whiteColour);

    ignoredErr := GetDialogItemAsControl(theDialogPtr, iClocks1, controlHdl);
    ignoredErr := GetControlData(controlHdl, kControlNoPart, kControlClockLongDateTag,
                        sizeof(longDateTimeStruc), Ptr(@longDateTimeStruc), actualSize);
    hour := longDateTimeStruc.hour;
    minute := longDateTimeStruc.minute;
    second := longDateTimeStruc.second;
    theString := 'Hour ';
    NumToString(SInt32(hour), tempString);
    theString := theString + tempString;
    theString := theString + ', Minute ';
    NumToString(SInt32(minute), tempString);
    theString := theString + tempString;
    theString := theString + ', Second ';
    NumToString(SInt32(second), tempString);
    theString := theString + tempString;
    MoveTo(100, 182);
    DrawString(theString);

    ignoredErr := GetDialogItemAsControl(theDialogPtr, iClocks2, controlHdl);
    ignoredErr := GetControlData(controlHdl, kControlNoPart, kControlClockLongDateTag,
                        sizeof(longDateTimeStruc), Ptr(@longDateTimeStruc), actualSize);
    day := longDateTimeStruc.day;
    month := longDateTimeStruc.month;
    year := longDateTimeStruc.year;
    theString := 'Day ';
    NumToString(SInt32(day), tempString);
    theString := theString + tempString;
    theString := theString + ', Month ';
    NumToString(SInt32(month), tempString);
    theString := theString + tempString;
    theString := theString + ', Year ';
    NumToString(SInt32(year), tempString);
    theString := theString + tempString;
    MoveTo(100, 196);
    DrawString(theString);
```

```
    ignoredErr := GetDialogItemAsControl(theDialogPtr, iClocks3, controlHdl);
    ignoredErr := GetControlData(controlHdl, kControlNoPart, kControlClockLongDateTag,
                              sizeof(longDateTimeStruc), Ptr(@longDateTimeStruc), actualSize);
    month := longDateTimeStruc.month;
    year := longDateTimeStruc.year;
    theString := 'Month ';
    NumToString(SInt32(month), tempString);
    theString := theString + tempString;
    theString := theString + ', Year ';
    NumToString(SInt32(year), tempString);
    theString := theString + tempString;
    MoveTo(100, 210);
    DrawString(theString);

    RGBBackColor(saveBackColour);
    SetPort(oldPort);
    end;
        { of procedure DoExtractDateTime }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ NumericFilter

function NumericFilter(controlHdl : ControlHandle;
                        var keyCode, charCode, modifiers : SInt16) : ControlKeyFilterResult;
    var
    returnValue : ControlKeyFilterResult;

    begin
    if ((char(charCode) >= '0') and (char(charCode) <= '9')) then
        begin
        NumericFilter := kControlKeyFilterPassKey;
        Exit(NumericFilter);
        end;

    case charCode of

        SInt16('-'), kLeftArrow, kRightArrow, kUpArrow, kDownArrow, kBackspace: begin
            returnValue := kControlKeyFilterPassKey;
            end;

        otherwise begin
            SysBeep(10);
            returnValue := kControlKeyFilterBlockKey;
            end;
        end;
            { of case statement }

    NumericFilter := returnValue;
    end;
        { of function NumericFilter }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ EditTextValidator

procedure EditTextValidator(controlHdl : ControlHandle);

    var
    oldText, newText : Str255;
    actualSize : Size;
    a : UInt8;
    count : UInt8;
    ignoredErr : OSErr;

    begin
    ignoredErr := GetControlData(controlHdl, kControlNoPart, kControlEditTextTextTag,
                    SInt16(oldText[0]), @oldText[1], actualSize);

    if (actualSize <= 255) then
        begin
        oldText[0] := char(actualSize);
        end
    else begin
        oldText[0] := char(255);
        end;

    count := 0;
    for a := 1 to SInt16(oldText[0]) do
        begin
        if (((('A' <= oldText[a]) and (oldText[a] <= 'Z')) or
                (('a' <= oldText[a]) and (oldText[a] <= 'z'))) or
                (oldText[a] = ' ') or (oldText[a] = '.')) then
```

```
            begin
            newText[count + 1] := oldText[a];
            count := count + 1;
            end;
        end;

    newText[0] := char(count);

    ignoredErr := SetControlData(controlHdl, kControlNoPart, kControlEditTextTextTag,
                SInt16(newText[0]), @newText[1]);

    Draw1Control(controlHdl);
    end;
        { of procedure EditTextValidator }

end.
    { of unit TabEditClock }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

{ ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
  GroupArrowsProgress.p
  ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ }

unit GroupArrowsProgress;


interface

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

uses

    { Other project files. }
    Controls3;

//
..........................................................................................................................................................................
........................... unit variables

var

gCacheSizeControlHdl : ControlHandle;
gLittleArrowsControlHdl : ControlHandle;
arrowsActionFunctionUPP : ControlActionUPP;

procedure DoGroupArrowsProgress;
procedure DoCheckBoxGroupBox(theDialogPtr : DialogPtr);
procedure DoPopupGroupBox(theDialogPtr : DialogPtr);
procedure DoAsynchronousAndProgress(theDialogPtr : DialogPtr);
procedure DoExtractCurrentStatus(theDialogPtr : DialogPtr);


implementation

uses

    { Other project files. }
    Callbacks;

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoGroupArrowsProgress

procedure DoGroupArrowsProgress;
    var
    theDialogPtr : DialogPtr;
    controlHdl : ControlHandle;
    eventFilterUPP : ModalFilterUPP;
    controlValue, itemHit : SInt16;
    theString : Str255;
    ignoredErr : OSErr;

    begin
    if (FrontWindow <> nil) then
        begin
        DoActivateWindow(FrontWindow, false);
        end;

    theDialogPtr := GetNewDialog(rGroupArrowsProgDialog, nil, WindowPtr(-1));
```

```
    if (theDialogPtr = nil) then
        begin
        ExitToShell;
        end;

    //
..........................................................................................................................................
......... set default button

    ignoredErr := SetDialogDefaultItem(theDialogPtr, kStdOkItemIndex);

    // ..................... create routine descriptors for event filter and little arrows action function

    eventFilterUPP := NewModalFilterProc(ProcPtr(@EventFilter));
    arrowsActionFunctionUPP := NewControlActionProc(ProcPtr(@ArrowsActionFunction));

    // ..................................................................................................... set initial value for checkbox
group box

    ignoredErr := GetDialogItemAsControl(theDialogPtr, iCheckboxGroup, controlHdl);
    SetControlValue(controlHdl, 1);

    // ..................................................................................... get and set initial cache value for little arrows

    ignoredErr := GetDialogItemAsControl(theDialogPtr, iLittleArrows, gLittleArrowsControlHdl);
    NumToString(GetControlValue(gLittleArrowsControlHdl), theString);
    theString := theString + 'K';
    ignoredErr := GetDialogItemAsControl(theDialogPtr, iStaticTextCache, gCacheSizeControlHdl);
    ignoredErr := SetControlData(gCacheSizeControlHdl, kControlNoPart,
                                    kControlStaticTextTextTag, SInt16(theString[0]), Ptr(@theString[1]));

    // ..................................................................................... hide second user pane in pop-up
group box

    ignoredErr := GetDialogItemAsControl(theDialogPtr, iUserPaneScoreAverage, controlHdl);
    HideControl(controlHdl);

    // ..................................................................................... show dialog and enter
ModalDialog loop

    ShowWindow(theDialogPtr);

    repeat
        begin
        ModalDialog(eventFilterUPP, itemHit);

        if (itemHit = iCheckboxGroup) then
            begin
            DoCheckBoxGroupBox(theDialogPtr);
            end
        else if(itemHit = iPopupGroup) then
            begin
            DoPopupGroupBox(theDialogPtr);
            end
        else if (itemHit = iCheckboxShowInitials) then
            begin
            ignoredErr := GetDialogItemAsControl(theDialogPtr, iCheckboxShowInitials, controlHdl);
            if (GetControlValue(controlHdl) <> 0) then
                begin
                controlValue := 0;
                end
            else begin
                controlValue := 1;
                end;
            SetControlValue(controlHdl, controlValue);
            end
        else if (itemHit = iCheckboxShowAverages) then
            begin
            ignoredErr := GetDialogItemAsControl(theDialogPtr, iCheckboxShowAverages, controlHdl);
            if (GetControlValue(controlHdl) <> 0) then
                begin
                controlValue := 0;
                end
            else begin
                controlValue := 1;
                end;
            SetControlValue(controlHdl, controlValue);
            end
        else if (itemHit = iDisclosureTriangle) then
            begin
```

```
            DoAsynchronousAndProgress(theDialogPtr);
            end
        else if (itemHit = iPushButtonExtract) then
            begin
            ignoredErr := GetDialogItemAsControl(theDialogPtr, iImageWell, controlHdl);
            Draw1Control(controlHdl);
            DoExtractCurrentStatus(theDialogPtr);
            end;
        end;
    until (itemHit = kStdOkItemIndex);

    // ..............................................................................................................................................................
........................................ clean up

    DisposeDialog(theDialogPtr);
    DisposeRoutineDescriptor(eventFilterUPP);
    DisposeRoutineDescriptor(arrowsActionFunctionUPP);

    gGroupArrowsProgressActive := false;
    end;
        { of procedure DoGroupArrowsProgress }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoCheckBoxGroupBox

procedure DoCheckBoxGroupBox(theDialogPtr : DialogPtr);
    var
    controlHdl : ControlHandle;
    controlValue : SInt16;
    ignoredErr : OSErr;

    begin
    ignoredErr := GetDialogItemAsControl(theDialogPtr, iCheckboxGroup, controlHdl);
    if (GetControlValue(controlHdl) <> 0) then
        begin
        controlValue := 0;
        end
    else begin
        controlValue := 1;
        end;
    SetControlValue(controlHdl, controlValue);

    if (controlValue = 0) then
        begin
        ignoredErr := GetDialogItemAsControl(theDialogPtr, iRadioGroupColour, controlHdl);
        ignoredErr := DeactivateControl(controlHdl);
        ignoredErr := GetDialogItemAsControl(theDialogPtr, iStaticTextColourDepth, controlHdl);
        ignoredErr := DeactivateControl(controlHdl);
        end
    else if (controlValue = 1) then
        begin
        ignoredErr := GetDialogItemAsControl(theDialogPtr, iRadioGroupColour, controlHdl);
        ignoredErr := ActivateControl(controlHdl);
        ignoredErr := GetDialogItemAsControl(theDialogPtr, iStaticTextColourDepth, controlHdl);
        ignoredErr := ActivateControl(controlHdl);
        end;
    end;
        { of procedure DoCheckBoxGroupBox }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoPopupGroupBox

procedure DoPopupGroupBox(theDialogPtr : DialogPtr);
    var
    controlHdl : ControlHandle;
    controlValue : SInt16;
    ignoredErr : OSErr;

    begin
    ignoredErr := GetDialogItemAsControl(theDialogPtr, iPopupGroup, controlHdl);
    controlValue := GetControlValue(controlHdl);

    if (controlValue = 1) then
        begin
        ignoredErr := GetDialogItemAsControl(theDialogPtr, iUserPaneScoreAverage, controlHdl);
        HideControl(controlHdl);
        ignoredErr := GetDialogItemAsControl(theDialogPtr, iUserPaneNamesInitials, controlHdl);
        ShowControl(controlHdl);
        end
    else if (controlValue = 2) then
        begin
```

```
      ignoredErr := GetDialogItemAsControl(theDialogPtr, iUserPaneNamesInitials, controlHdl);
      HideControl(controlHdl);
      ignoredErr := GetDialogItemAsControl(theDialogPtr, iUserPaneScoreAverage, controlHdl);
      ShowControl(controlHdl);
      end;
   end;
      { of procedure DoPopupGroupBox }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoAsynchronousAndProgress

procedure DoAsynchronousAndProgress(theDialogPtr : DialogPtr);
   var
   controlHdl : ControlHandle;
   controlValue : SInt16;
   ditlHdl : Handle;
   indeterminateFlag : boolean;
   expandString : Str255;
   collapseString : Str255;
   ignoredErr : OSErr;

   begin
   indeterminateFlag := true;
   expandString   := 'Hide Progress Indicator and Asynchronous Arrows';
   collapseString := 'Show Progress Indicator and Asynchronous Arrows';

   ignoredErr := GetDialogItemAsControl(theDialogPtr, iDisclosureTriangle, controlHdl);

   if (GetControlValue(controlHdl) <> 0) then
      begin
      controlValue := 0;
      end
   else begin
      controlValue := 1;
      end;
   SetControlValue(controlHdl, controlValue);

   if (controlValue = 1) then
      begin
      ditlHdl := GetResource('DITL', 130);
      AppendDITL(theDialogPtr, ditlHdl, appendDITLBottom);
      ReleaseResource(ditlHdl);

      ignoredErr := GetDialogItemAsControl(theDialogPtr, iProgressBar, controlHdl);
      ignoredErr := SetControlData(controlHdl, kControlNoPart,
                                kControlProgressBarIndeterminateTag, sizeof(indeterminateFlag),
                                Ptr(@indeterminateFlag));

      ignoredErr := GetDialogItemAsControl(theDialogPtr, kStdOkItemIndex, controlHdl);
      MoveControl(controlHdl, 267, 336);

      ignoredErr := GetDialogItemAsControl(theDialogPtr, iStaticTextDisclosure, controlHdl);
      ignoredErr := SetControlData(controlHdl, kControlNoPart, kControlStaticTextTextTag,
                             SInt16(expandString[0]), Ptr(@expandString[1]));
      Draw1Control(controlHdl);

      end
   else if (controlValue = 0) then
      begin
      ignoredErr := GetDialogItemAsControl(theDialogPtr, kStdOkItemIndex, controlHdl);
      MoveControl(controlHdl, 267, 259);

      SizeWindow(theDialogPtr, 348, 302, false);
      ignoredErr := GetDialogItemAsControl(theDialogPtr, iStaticTextDisclosure, controlHdl);
      ignoredErr := SetControlData(controlHdl, kControlNoPart, kControlStaticTextTextTag,
                             SInt16(collapseString[0]), Ptr(@collapseString[1]));
      Draw1Control(controlHdl);
      end;
   end;
      { of procedure DoAsynchronousAndProgress }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoExtractCurrentStatus

procedure DoExtractCurrentStatus(theDialogPtr : DialogPtr);
   var
   oldPort : GrafPtr;
   saveBackColour, whiteColour : RGBColor;
   controlHdl : ControlHandle;
   controlValue : SInt16;
   theString : Str255;
   ignoredErr : OSErr;
```

```
begin
whiteColour.red := $FFFF;
whiteColour.green := $FFFF;
whiteColour.blue := $FFFF;

GetPort(oldPort);
SetPort(theDialogPtr);
GetBackColor(saveBackColour);
RGBBackColor(whiteColour);

ignoredErr := GetDialogItemAsControl(theDialogPtr, iCheckboxGroup, controlHdl);
controlValue := GetControlValue(controlHdl);
MoveTo(98, 191);
if (controlValue <> 0) then
    begin
    theString := 'Use colour, ';
    ignoredErr := GetDialogItemAsControl(theDialogPtr, iRadioGroupColour, controlHdl);
    controlValue := GetControlValue(controlHdl);
    if (controlValue = 1) then
        begin
        theString := theString + ' 8 bit.';
        end
    else if (controlValue = 2) then
        begin
        theString := theString + ' 16 bit.';
        end
    else if (controlValue = 3) then
        begin
        theString := theString + ' 32 bit.';
        end;
    end
else begin
    theString := 'Don''t use colour.';
    end;

DrawString(theString);

ignoredErr := GetDialogItemAsControl(theDialogPtr, iPopupGroup, controlHdl);
controlValue := GetControlValue(controlHdl);
MoveTo(98, 205);
if (controlValue = 1) then
    begin
    theString := 'Player, ';
    ignoredErr := GetDialogItemAsControl(theDialogPtr, iRadioGroupNames, controlHdl);
    controlValue := GetControlValue(controlHdl);
    if (controlValue = 1) then
        begin
        theString := theString + 'name first, ';
        end
    else if (controlValue = 2) then
        begin
        theString := theString + 'name last, ';
        end;
    ignoredErr := GetDialogItemAsControl(theDialogPtr, iCheckboxShowInitials, controlHdl);
    controlValue := GetControlValue(controlHdl);
    if (controlValue = 1) then
        begin
        theString := theString + ' show number.';
        end
    else if (controlValue = 0) then
        begin
        theString := theString + ' no number.';
        end;
    end
else if (controlValue = 2) then
    begin
    theString := 'Score, ';
    ignoredErr := GetDialogItemAsControl(theDialogPtr, iRadioGroupScores, controlHdl);
    controlValue := GetControlValue(controlHdl);
    if (controlValue = 1) then
        begin
        theString := theString + 'batting, ';
        end
    else if (controlValue = 2) then
        begin
        theString := theString + 'bowling, ';
        end;
    ignoredErr := GetDialogItemAsControl(theDialogPtr, iCheckboxShowAverages, controlHdl);
    controlValue := GetControlValue(controlHdl);
```

```
        if (controlValue = 1) then
            begin
            theString := theString + 'show average.';
            end
        else if (controlValue = 0) then
            begin
            theString := theString + 'no average.';
            end;
        end;

    DrawString(theString);

    MoveTo(98, 219);
    DrawString('Cache size: ');
    ignoredErr := GetDialogItemAsControl(theDialogPtr, iLittleArrows, controlHdl);
    NumToString(GetControlValue(controlHdl), theString);
    DrawString(theString);

    RGBBackColor(saveBackColour);
    SetPort(oldPort);
    end;
        { of procedure DoExtractCurrentStatus }

end.
    { of unit GroupArrowsProgress }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

{ ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
  Sliders.p
  ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ }

unit Sliders;


interface

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

uses

    { Other project files. }
    Controls3;

// ..................................................................................................................................................................................................
........................... unit variables

var

sliderActionFunction1UPP : ControlActionUPP;
sliderActionFunction2UPP : ControlActionUPP;
userPaneDrawFunctionUPP : ControlUserPaneDrawUPP;
userPaneActivateFunctionUPP : ControlUserPaneActivateUPP;
gSlider1Hdl : ControlHandle;
gSlider2Hdl : ControlHandle;
gSlider3Hdl : ControlHandle;
gSlider4Hdl : ControlHandle;
gSlider5Hdl : ControlHandle;
gSlider6Hdl : ControlHandle;


procedure DoSliderUserPane;
procedure DoDrawSliderValues(theDialogPtr : DialogPtr; controlHdl : ControlHandle);
procedure UserPaneDrawFunction(controlHdl : ControlHandle; thePart : SInt16);
procedure UserPaneActivateFunction(controlHdl : ControlHandle; activating : boolean);


implementation

uses

    { Other project files. }
    Callbacks;

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoSliderUserPane

procedure DoSliderUserPane;
```

```
    var
    dialogPtr : DialogPtr;
    eventFilterUPP : ModalFilterUPP;
    controlHdl : ControlHandle;
    itemHit : SInt16;
    ignoredErr : OSErr;

    begin
    if (FrontWindow <> nil) then
        begin
        DoActivateWindow(FrontWindow, false);
        end;

    dialogPtr := GetNewDialog(rSlidersDialog, nil, WindowPtr(-1));
    if (dialogPtr = nil) then
        begin
        ExitToShell;
        end;

    //
..........................................................................................................................................
......... set default button

    ignoredErr := SetDialogDefaultItem(dialogPtr, kStdOkItemIndex);

    // ..... create routine descriptor for event filter function

    eventFilterUPP := NewModalFilterProc(ProcPtr(@EventFilter));

    // ..... create routine descriptors for slider action functions

    sliderActionFunction1UPP := NewControlActionProc(@SliderActionFunction1);
    sliderActionFunction2UPP := NewControlActionProc(@SliderActionFunction2);

    // ..... create routine descriptors for user pane functions, set user pane functions

    userPaneDrawFunctionUPP := NewRoutineDescriptor(@UserPaneDrawFunction,
                                                    uppControlUserPaneDrawProcInfo, GetCurrentISA);
    ignoredErr := GetDialogItemAsControl(dialogPtr, iUserPane1, controlHdl);
    ignoredErr := SetControlData(controlHdl, kControlNoPart, kControlUserPaneDrawProcTag,
                            sizeof(userPaneDrawFunctionUPP), Ptr(@userPaneDrawFunctionUPP));

    userPaneActivateFunctionUPP := NewRoutineDescriptor(@UserPaneActivateFunction,
                                                        uppControlUserPaneActivateProcInfo, GetCurrentISA);
    ignoredErr := GetDialogItemAsControl(dialogPtr, iUserPane1, controlHdl);
    ignoredErr := SetControlData(controlHdl, kControlNoPart, kControlUserPaneActivateProcTag,
                            sizeof(userPaneActivateFunctionUPP), Ptr(@userPaneActivateFunctionUPP));

    // ..... get control handles of, and draw initial control values for, top four sliders

    ignoredErr := GetDialogItemAsControl(dialogPtr, iSlider1, gSlider1Hdl);
    DoDrawSliderValues(dialogPtr, gSlider1Hdl);
    ignoredErr := GetDialogItemAsControl(dialogPtr, iSlider2, gSlider2Hdl);
    DoDrawSliderValues(dialogPtr, gSlider2Hdl);
    ignoredErr := GetDialogItemAsControl(dialogPtr, iSlider3, gSlider3Hdl);
    DoDrawSliderValues(dialogPtr, gSlider3Hdl);
    ignoredErr := GetDialogItemAsControl(dialogPtr, iSlider4, gSlider4Hdl);
    DoDrawSliderValues(dialogPtr, gSlider4Hdl);

    // ..... get control handles and values for bottom two sliders, set colour

    ignoredErr := GetDialogItemAsControl(dialogPtr, iSlider5, gSlider5Hdl);
    gRedColour.red := 2 * GetControlValue(gSlider5Hdl);
    ignoredErr := GetDialogItemAsControl(dialogPtr, iSlider6, gSlider6Hdl);
    gBlueColour.blue := 2 * GetControlValue(gSlider5Hdl);

    // ..... show dialog and enter
ModalDialog loop

    ShowWindow(dialogPtr);

    repeat
        begin
        ModalDialog(eventFilterUPP, itemHit);
        end;
    until (itemHit = kStdOkItemIndex);

    //
..........................................................................................................................................
..... clean up
```

```
    DisposeDialog(dialogPtr);
    DisposeRoutineDescriptor(eventFilterUPP);
    DisposeRoutineDescriptor(sliderActionFunction1UPP);
    DisposeRoutineDescriptor(sliderActionFunction2UPP);
    DisposeRoutineDescriptor(userPaneDrawFunctionUPP);
    DisposeRoutineDescriptor(userPaneActivateFunctionUPP);
    gSlidersActive := false;
    end;
        { of procedure DoSliderUserPane }

//
.............................................................................................................................................................
.............. DoDrawSliderValues

procedure DoDrawSliderValues(theDialogPtr : DialogPtr; controlHdl : ControlHandle);
    var
    theString : Str255;
    staticTextItem : SInt16;
    ignoredErr : OSErr;

    begin
    NumToString(GetControlValue(controlHdl), theString);

    if (controlHdl = gSlider1Hdl) then
        begin
        staticTextItem := iSlider1StaticText;
        end
    else if (controlHdl = gSlider2Hdl) then
        begin
        staticTextItem := iSlider2StaticText;
        end
    else if (controlHdl = gSlider3Hdl) then
        begin
        staticTextItem := iSlider3StaticText;
        end
    else if (controlHdl = gSlider4Hdl) then
        begin
        staticTextItem := iSlider4StaticText;
        end;

    ignoredErr := GetDialogItemAsControl(theDialogPtr, staticTextItem, controlHdl);
    ignoredErr := SetControlData(controlHdl, kControlNoPart, kControlStaticTextTextTag,
            SInt16(theString[0]), Ptr(@theString[1]));
    Draw1Control(controlHdl);
    end;
        { of procedure DoDrawSliderValues }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ UserPaneDrawFunction

procedure UserPaneDrawFunction(controlHdl : ControlHandle; thePart : SInt16);
    var
    theRect : Rect;
    ignoredErr : OSErr;

    begin
    SetRect(theRect, 225, 186, 239, 200);
    ignoredErr := DrawThemeGenericWell(theRect, SInt16(gDrawActivated), true);

    if gDrawActivated then
        begin
        RGBForeColor(gRedColour);
        PaintRect(theRect);
        end;

    SetRect(theRect, 225, 207, 239, 221);
    ignoredErr := DrawThemeGenericWell(theRect, SInt16(gDrawActivated), true);

    if gDrawActivated then
        begin
        RGBForeColor(gBlueColour);
        PaintRect(theRect);
        end;
    end;
        { of procedure UserPaneDrawFunction }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ UserPaneActivateFunction

procedure UserPaneActivateFunction(controlHdl : ControlHandle; activating : boolean);
    begin
```

```
    gDrawActivated := activating;
    end;
        { of procedure UserPaneActivateFunction }

end.
    { of unit Sliders }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

{ ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
  TextBox.c
  ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ }

unit TextBox;


interface

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

uses

    { Other project files. }
    Controls3;

// ..................................................................................................................................................................
........................... unit variables

 procedure DoAboutBox;

implementation

uses

    { Other project files. }
    Callbacks;

// ..................................................................................................................................................................
...................................... DoAboutBox

procedure DoAboutBox;
    var
    dialogPtr : DialogPtr;
    eventFilterUPP : ModalFilterUPP;
    itemHit : SInt16;
    ignoredErr : OSErr;

    begin
    if (FrontWindow <> nil) then
        begin
        DoActivateWindow(FrontWindow, false);
        end;

    dialogPtr := GetNewDialog(rAboutDialog, nil, WindowPtr(-1));
    if (dialogPtr = nil) then
        begin
        ExitToShell;
        end;

    SetWTitle(dialogPtr,'About Box (Text Boxes)');

    //
..................................................................................................................................................................
......... set default button

    ignoredErr := SetDialogDefaultItem(dialogPtr, kStdOkItemIndex);

    // ........................................................................................................................ create routine descriptor for event
filter

    eventFilterUPP := NewModalFilterProc(ProcPtr(@EventFilter));

    // ................................................................................................................................. show dialog and enter modal
dialog loop

    ShowWindow(dialogPtr);
```

```
    repeat
        begin
        ModalDialog(eventFilterUPP, itemHit);
        end;
    until (itemHit = kStdOkItemIndex);

    //
.......................................................................................................................................................................
...................................... clean up

    DisposeDialog(dialogPtr);
    DisposeRoutineDescriptor(eventFilterUPP);
    end;
        { of procedure DoAboutBox }
end.
    { of unit TextBox }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

{ ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
  Callbacks.p
  ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ }

unit Callbacks;


interface

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

uses

    { Universal Interfaces. }
    MacTypes, Dialogs, Events,

    { Other project files. }
    Controls3;

function EventFilter(thetheDialogPtr  : DialogPtr; {const} var theEvent : EventRecord;
                        var itemHit : SInt16) : boolean;
procedure ArrowsActionFunction(controlHdl : ControlHandle; partCode : SInt16);
procedure SliderActionFunction1(controlHdl : ControlHandle; partCode : SInt16);
procedure SliderActionFunction2(controlHdl : ControlHandle; partCode : SInt16);


implementation

uses

    { Other project files. }
    GroupArrowsProgress, Sliders;

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ EventFilter

function EventFilter(theDialogPtr  : DialogPtr; {const} var theEvent : EventRecord;
                        var itemHit : SInt16) : boolean;

    var
    handledEvent : boolean;
    oldPort : GrafPtr;
    mouseXY : Point;
    controlHdl : ControlHandle;
    ignoredCode : ControlPartCode;

    begin
    handledEvent := false;

    if ((theEvent.what = updateEvt) and (WindowPtr(theEvent.message) <> theDialogPtr)) then
        begin
        DoUpdate(theEvent);
        end
    else begin
        GetPort(oldPort);
        SetPort(theDialogPtr);

        if gGroupArrowsProgressActive then
            begin
```

```
         if (theEvent.what = mouseDown) then
            begin
            mouseXY := Point(theEvent.where);
            GlobalToLocal(mouseXY);
            if (FindControl(mouseXY, theDialogPtr, controlHdl) <> 0) then
               begin
               if (controlHdl = gLittleArrowsControlHdl) then
                  begin
                  ignoredCode := TrackControl(controlHdl, mouseXY, arrowsActionFunctionUPP);
                  handledEvent := true;
                  end;
               end;
            end;
         end
      else if gSlidersActive then
         begin
         if (theEvent.what = mouseDown) then
            begin
            mouseXY := theEvent.where;
            GlobalToLocal(mouseXY);
            if( FindControl(mouseXY, theDialogPtr, controlHdl) <> 0) then
               begin
               if ((controlHdl = gSlider1Hdl) or (controlHdl = gSlider2Hdl)) then
                  begin
                  ignoredCode := TrackControl(controlHdl, mouseXY, nil);
                  DoDrawSliderValues(theDialogPtr, controlHdl);
                  handledEvent := true;
                  end
               else if ((controlHdl = gSlider3Hdl) or (controlHdl = gSlider4Hdl)) then
                  begin
                  ignoredCode := TrackControl(controlHdl, mouseXY, sliderActionFunction1UPP);
                  handledEvent := true;
                  end
               else if ((controlHdl = gSlider5Hdl) or (controlHdl = gSlider6Hdl)) then
                  begin
                  ignoredCode := TrackControl(controlHdl, mouseXY, sliderActionFunction2UPP);
                  handledEvent := true;
                  end;
               end;
            end;
         end
      else begin
         handledEvent := StdFilterProc(theDialogPtr, theEvent, itemHit);
         end;

      SetPort(oldPort);
      end;

   EventFilter := handledEvent;
   end;
      { of function EventFilter }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ ArrowsActionFunction

procedure ArrowsActionFunction(controlHdl : ControlHandle; partCode : SInt16);
   var
   theString : Str255;
   controlValue : SInt32;
   ignoredErr : OSErr;

   begin
   if (partCode <> 0) then
      begin
      controlValue := GetControlValue(controlHdl);

      case partCode of

         kControlUpButtonPart: begin
            controlValue := controlValue + 32;
            if (controlValue > GetControlMaximum(controlHdl)) then
               begin
               controlValue := GetControlMaximum(controlHdl);
               return;
               end;
            end;

         kControlDownButtonPart: begin
            controlValue := controlValue - 32;
            if (controlValue < GetControlMinimum(controlHdl)) then
               begin
```

```
                controlValue := GetControlMinimum(controlHdl);
                return;
                end;
            end;

        otherwise begin
            end;
        end;
            { of case statement }

    SetControlValue(controlHdl, controlValue);

    NumToString(controlValue, theString);
    theString := theString + 'K';
    ignoredErr := SetControlData(gCacheSizeControlHdl, kControlNoPart,
                                kControlStaticTextTextTag, SInt16(theString[0]), Ptr(@theString[1]));
    Draw1Control(gCacheSizeControlHdl);
    end;
  end;
    { of procedure ArrowsActionFunction }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ SliderActionFunction1

procedure SliderActionFunction1(controlHdl : ControlHandle; partCode : SInt16);
  var
  staticTextItem : SInt16;
  theString : Str255;
  theDialogPtr : DialogPtr;
  ignoredErr : OSErr;

  begin
  NumToString(SInt32(GetControlValue(controlHdl)), theString);

  theDialogPtr := controlHdl^^.contrlOwner;

  if (controlHdl = gSlider3Hdl) then
    begin
    staticTextItem := iSlider3StaticText;
    end
  else if (controlHdl = gSlider4Hdl) then
    begin
    staticTextItem := iSlider4StaticText;
    end;

  ignoredErr := GetDialogItemAsControl(theDialogPtr, staticTextItem, controlHdl);
  ignoredErr := SetControlData(controlHdl, kControlNoPart, kControlStaticTextTextTag,
                SInt16(theString[0]), Ptr(@theString[1]));
  Draw1Control(controlHdl);
  end;
    { of procedure SliderActionFunction1 }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ SliderActionFunction2

procedure SliderActionFunction2(controlHdl : ControlHandle; partCode : SInt16);
  var
  controlValue : SInt16;
  theRect : Rect;

  begin
  controlValue := GetControlValue(controlHdl);

  if (controlHdl = gSlider5Hdl) then
    begin
    gRedColour.red := 2 * controlValue;
    RGBForeColor(gRedColour);
    SetRect(theRect, 225, 186, 239, 200);
    end
  else if (controlHdl = gSlider6Hdl) then
    begin
    gBlueColour.blue := 2 * controlValue;
    RGBForeColor(gBlueColour);
    SetRect(theRect, 225, 207, 239, 221);
    end;

  PaintRect(theRect);
  end;
    { of procedure SliderActionFunction2 }

end.
  { of unit Callbacks }
```

{ ◊◊ }

Demonstration Program Comments

When this program is run, the user should:

- Choose the About Controls3… item from the Apple menu to view an About dialog containing a standard Text Box control and an auto-scrolling Text Box control. (Note: If Mac OS 8.5 is not present, the About Controls3… item will be disabled and this action will not be possible.)
- Choose items from the Demonstration menu to view and operate the various controls.
- Choose Show Balloons from the Help menu and note the information (and, in some cases, the instructions) in the help balloons as the cursor is moved over the various controls in the window and dialog boxes.
- Click in the Finder and then back in the window/dialog box, noting control activation/deactivation.
- In the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window:
    - Click in the various controls, noting in the window header the control part code returned by FindControl (or immediately by TrackControl in the case of the non-tracking icon controls).
    - Click in the various controls and then release the mouse button both within and outside the control, noting the control part code returned by TrackControl.
- In the Tab, Edit Text, and Clock Controls dialog box:
    - In the Edit Text Controls tab:
        - Change the keyboard focus using the tab key and mouse clicks.
        - Enter a name, age, and password, noting the effect of the key filter function attached to the "Age" edit text control and the behaviour of the "Password" edit text control.
        - Paste some text containing characters other than alphabetic characters, the space character, and the period character to the "Name" edit text control, noting that the edit text validation function strips out the "illegal" characters.
        - Note the cursor shape change when the cursor is over the top two edit text controls.
        - Click the Extract push button to extract and display the contents of the edit text controls.
- In the Group Boxes, Arrows, and Progress Bar dialog box:
    - Change the group box settings, the settings of the controls within the group boxes, and the (simulated) cache size setting controlled by the little arrows, and then click the Extract button to extract and display the settings of the various controls.
    - Click the disclosure triangle to show and hide the asynchronous arrows and indeterminate progress bar.
- In the Sliders and User Pane Functions dialog:
    - Operate the sliders and note the difference in the appearance of the dragged indicator in the live scrolling and non-live scrolling variants of the sliders.
    - Note the appearance, in the activated and deactivated modes, of the two custom "controls" to the right of the two horizontal sliders. Also note that these controls are re-drawn, in the appropriate mode, when an overlaying window or help balloon is removed.

Control3Program.p

Controls3Program.p is simply the basic "engine" which supports the demonstration.

The Gestalt call determines whether Mac OS 8.5 or later is present. If so, the About Controls 3… item in the Apple menu is disabled. (The dialog box opened when this item is chosen contains Text Box controls, the CDEFs for which are only included with Mac OS 8.5 and later.)

Controls3.p

constants

After the usual constants relating to the menu bar resource, menu resources and IDs, and menu items are established, constants are established for the resource IDs and dialog item numbers associated with the various demonstrations.

type definitions

A variable of type BevelDocRecord will be used to hold handles to the various controls created in the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window.

Global Variables

gBevelAndImageActive will be set to true while the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window is open. gGroupArrowsProgressActive will be set to true while the Group Boxes, Arrows, and Progress Bar dialog box is open. gSlidersActive will be set to true while the Sliders and User Pane Functions dialog is open.

DoMouseDown

In the inContent case, if a mouse-down occurs in the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window when it is the front window, the routine DoBevelImagePictIconContent is called.

In the inGoAway case, the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window is disposed of and the global variable gBevelAndImageActive is set to false.

DoMenuChoice

In the mDemonstration case, the relevant application-defined routine is called when the user chooses items in the Demonstration menu. In three cases, the relevant global variable is also set to true.

DoUpdate

If the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window is open, UpdateControls is called to redraw the controls in the appropriate mode. Also, two application-defined routines are called to draw the current window header string and the legends in the appropriate mode.

DoActivateWindow

If the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window is open, and if it receives an activate event, GetRootControl is called to get a handle to the root control created for the window by the application (see below). The controls are then activated or deactivated en masse, and the window header string and the legends are re-drawn in the appropriate mode.

BevelImagePictIcon.p

BevelImagePictIcon.p contains most of the source code relating to the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window.

DoBevelImagePictIcon

DoBevelImagePictIcon opens a window, creates a relocatable block for a structure of type BevelDocStruc (to whose fields handles to the various controls will be assigned) and assigns the handle to this block to the refCon field of the window structure.

First, some initial advisory text for display in the window header is assigned to a global variable. The call to GetNewCWindow then creates the window. (Note that error handling here and in other areas of the demonstration is somewhat rudimentary in that the program simply terminates.) The window's graphics port is set as the current port for drawing and the window's text size is set to 10pt.

The call to SetThemeWindowBackground sets an Appearance-compliant colour/pattern for the window's content area. NewHandle creates the block for the variable of type BevelDocStruc and SetWRefCon assigns the handle to the block to the window structure's refCon field.

Before ShowWindow is called to make the window visible, the application-defined routine doGetControls is called to create the controls. The call to DoGetDepthAndDevice assigns appropriate values to two global variables required in later calls to the Appearance Manager function SetThemeTextColor.

DoGetControls

DoGetControls creates the controls from the various 'CNTL' resources.

At the first line, the root control is created. The first control created must be always be the root control (which is implemented as a user pane).

A handle to the structure in which the handles to the control structures will be stored is then retrieved. The following calls to GetNewControl create a control structure for each control, insert the structure into the control list for the specified window and draw the control. At the same time, the handle to each control is assigned to the appropriate field of the window's "document" structure.

At the next block, SetControlData is twice called with the kControlBevelButtonTextPlaceTag control data tag constant. The bevel button text placement constant passed in these calls causes the text in the specified bevel buttons to be placed above the graphic.

At the next block, SetControlData is called with the kControlBevelButtonCenterPopupGlyphTag control data tag constant to cause the pop-up glyph in the specified bevel button to be centred.

At the next block, the font for the text in the specified bevel button is changed. The flags and font fields of a control font style structure are assigned constants so that the following call to SetControlFontStyle will set the font to the small emphasised system font.

The final line sets the value of the specified bevel button to 2, causing it to appear in the mixed state.

DoBevelImagePictIconContent

Recall that, if a mouse-down occurs in the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window when it is the front window, DoBevelImagePictIconContent is called to further handle the event.

At the first line a handle to the structure containing handles to all the controls is obtained.

The call to GlobalToLocal changes the coordinates at which the mouse-down occurred to the local coordinates required by the following call to FindControl. The constant representing the part code returned by FindControl is then drawn in the window header.

If the part code returned by FindControl is not kControlNoPart (0), meaning that an enabled control was under the mouse cursor at the time of the mouse-down, the if block executes. TrackControl takes control while the mouse button remains down, returning a part code when the button is released.

In the case of the first 10 specified controls (bevel buttons), the control part code returned by TrackControl is simply drawn in the window header.

The next four controls are bevel button with menus. In these cases, if the part code returned indicates that the mouse button was released while the cursor was within the menu, GetControlData is called with the kControlBevelButtonMenuValueTag tag constant to get the menu item number. The part code and menu item number are then drawn in the window header. If the cursor was not in the menu when the mouse button was released, the part code returned by TrackControl is drawn in the window header.

The 10th, 11th, and 12th bevel buttons have toggle behaviour. With some assistance from the application, they behave like radio buttons. If the control in which the mouse-down occurred was one of these bevel buttons, and if the cursor was still within the control when the mouse button was released, SetControlValue is called to set the three controls to the unchecked state, following which the selected control is set to the checked state. Either way, the part code returned by TrackControl is drawn in the window header.

The 18th bevel button is used to demonstrate bevel button graphic alignments using a single bevel button. A mouse-down within this button causes the application-defined routine doGraphicAlignment to be called.

The 19th bevel button is used to demonstrate bevel button text alignments using a single bevel button. A mouse-down within this button causes the application-defined routine doTextAlignment to be called.

The 20th bevel button is used to demonstrate text offsetting using a single bevel button. A mouse-down within this button causes the application-defined routine doTextOffset to be called.

The 21st bevel button is used to demonstrate bevel button text placement in relation to the bevel button's graphic using a single bevel button. A mouse-down within this button causes the application-defined routine doTextPlacement to be called.

Finally, if the mouse-down was in an image well, picture control, or icon control, and except for the non-tracking picture and icon control variants, the part code returned by TrackControl is drawn in the window header.

DoDrawPartCode

DoDrawPartCode takes part codes and menu item numbers and assembles them into descriptive strings for drawing in the window header. The constants used are stored in 'STR#' resources, and are retrieved using GetIndString. In the final two lines, Draw1Control is called to redraw the window header control (in effect erasing the current text drawn within the window header area) and doDrawMessage is called to draw the assembled string.

DoGraphicAlignment

DoGraphicAlignment is called to demonstrate the effect of all bevel button graphic alignment constants except kControlBevelButtonAlignSysDirection. Each time around the for loop, the alignment constant is changed before SetControlData is called with the kControlBevelButtonGraphicAlignTag tag constant to set the alignment, Draw1Control is called to redraw the bevel button, and the alignment constant used during that pass is drawn in the window header.

DoTextAlignment

DoTextAlignment is called to demonstrate the effect of all bevel button text alignment constants except kControlBevelButtonAlignTextSysDirection. Each time around the for loop, the alignment constant is changed before SetControlData is called with the kControlBevelButtonTextAlignTag tag constant to set the alignment, Draw1Control is called to redraw the bevel button, and the alignment constant used during that pass is drawn in the window header.

DoTextOffset

DoTextOffset is called to demonstrate text offsetting from the left and the right within a bevel button. For the purposes of the demonstration, two animations involving incrementing offsets values are used. Prior to first animation, SetControlData is called with the kControlBevelButtonTextOffsetTag tag to align the text on the left. Prior to second animation, SetControlData is called with the kControlBevelButtonTextOffsetTag tag to align the text on the right. Within the for loops, SetControlData is called with the kControlBevelButtonTextOffsetTag tag to offset the text from the left or right by the specified number of pixels, and Draw1Control re-draws the control.

DoTextPlacement

DoTextPlacement is called to demonstrate the effect of all bevel button text placement constants except kControlBevelButtonPlaceSysDirection. Within a for loop which increments the text placement constant, SetControlData is called with the kControlBevelButtonTextPlaceTag tag and Draw1Control re-draws the control.

DoDrawMessage and doDrawLegends

DoDrawMessage and doDrawLegends are incidental to the demonstration. Both functions draw text in the window in an Appearance-compliant colour which depends on whether the window is currently in front or in the background.

TabEditClock.p

TabEditClock.p creates a movable modal dialog in which is demonstrated a tab control, edit text controls and clock controls. An application-defined numeric key filter function is attached to the second edit text control. The third edit text control is for password input. The controls displayed by each tab are embedded in user panes.

The kDialogFlagsUsesControlHierarchy flag is set in the 'dlgx' resources for all dialogs used in the demonstration. Recall that this means that the Dialog Manager creates a root control in the dialog and establishes an embedding hierarchy, that all dialog items automatically become controls, and that the Dialog Manager uses AutoEmbedControl to position dialog items in an embedding hierarchy based on both visual containment and their item list number.

The dialog item list used by the dialog created by TabEditClock.p is shown in the following, in which the indentation represents the embedding hierarchy:

1. OK push button primitive.
2. Tab control (kTabControlLargeProc variant).
   Visually contains the two user panes (items 3 and 12) and thus automatically emebeds those items.
   3. User pane control with kControlSupportsEmbedding feature bit set (initial value set to 2).
      Visually contains items 4 to 11 and thus automatically embeds those items.
      4. Static text primitive "Name:".
      5 Edit text primitive.
      6. Static text primitive "Age:".
      7. Edit text primitive.
      8. Static text primitive "Password:".
      9. Edit text control (kControlEditTextPasswordProc variant).
      10. Extract push button primitive.
      11. Image well control.
   12. User pane control with kControlSupportsEmbedding feature bit set (initial value set to 2).
      Visually contains items 13 to 20 and thus automatically embeds those items.
      13. Static text primitive "Hours, Mins, Secs:".
      14. Clock control (kControlClockTimeSecondsProc variant).
      15. Static text primitive "Date, Month, Year:".
      16. Clock control (kControlClockDateProc variant).
      17. Static text primitive "Month, Year:".
      18. Clock control (kControlClockMonthYearProc variant).
      19. Extract push button primitive.
      20. Image well control.
21. Clock control (kControlClockTimeSecondsProc variant, live, display only.)
22. Group box (kControlGroupBoxSecondaryTextTitleProc variant).
   Visually contains item 23 and thus automatically embeds that item:
   23. Static text primitive.

DoTabEditClock

DoTabEditClock creates the dialog, calls ModalDialog to handle events in the dialog, and disposes of the dialog when the user hits the OK push button.

Firstly, and as is always required when a dialog box is to be displayed, the front window (if one exists) is explicitly deactivated.

The call to GetNewDialog creates the dialog. The call to SetWTitle sets the window's title so as to force an association with the 'hrct' resource containing the help balloons for the Edit Text Field tab. (Later calls to SetWTitle in this function are used for similar purposes.)

SetDialogDefaultItem tells the Dialog Manager which is the default push button item, to alias the Return and Enter keys to that item, and to draw the default ring around that item. SetDialogTracksCursor will cause the Dialog Manager to change the cursor shape to the I-Beam cursor whenever the cursor is over an edit text control specified in the 'DITL' as an edit text primitive.

The first call to GetDialogItemAsControl gets a handle to the user pane used for the clocks tab. HideControls hides this user pane and all the controls automatically embedded within it.

At the next block, SetControlData is called with the kControlEditTextTextTag tag to assign some text to the first edit text control. The fields of an edit text selection structure are then assigned values which will cause the following call to SetControlData to select the entire edit text control.

The next block creates routine descriptors for application-defined event filter, key filter, and edit text validator functions. (The event filter function is defined in the source code file Callbacks.c.) SetControlData is then called with the kControlEditTextValidationProcTag tag to attach the edit text validation function to the first edit text control. SetControlData is then called again with the kControlEditTextKeyFilterTag tag to attach the key filter function to the second edit text control.

With those preliminaries attended to, ShowWindow is called to display the window, following which the ModalDialog loop is entered. The loop will execute until the user hits the OK push button. Note that the previously created UPP is passed in the filterProc parameter of ModalDialog.

When a mouse-down event occurs in an enabled item, ModalDialog returns the item number of the item hit. If the item hit was the tab item, GetDialogItemAsControl is called to get a handle to the tab control and GetControlValue is called to determine which of the two tabs was hit. If the tab hit was the Edit Text Fields tab, the Clock Controls user pane is hidden, the Edit Text Fields user pane is shown, and the keyboard focus is set to the first edit text field. If the tab hit was the Clock Controls tab, the Edit Text Fields user pane is hidden, the Clock Controls user pane is shown, and the keyboard focus is set to the first clock control.

If the item hit was the Extract push button in the Edit Text Fields pane, the image well control is re-drawn to erase any previous text and an application-defined routine is called to extract and display the contents of the edit text controls.

If the item hit was the Extract push button in the Clock Controls pane, the image well control is re-drawn to erase any previous text and an application-defined routine is called to extract and display the contents of the clock controls.

When the user hits the OK button, DisposeDialog closes the dialog and disposes of the associated memory, and the routine descriptors associated with the event filter function and key filter function are disposed of.

DoExtractEditText

DoExtractEditText extracts the text from the edit text controls and draws it in the image well area. In the case of the Password edit text control, the text extracted is the actual password typed, not the bullet text.

In the case of the first two edit text controls, GetDialogItem is used to get the handle to the hText field of the TERec structure used by the item. This is where the characters are located. (Exactly what is returned in the iHandle field of a GetDialogItem call depends on the item type.) GetDialogItemText then copies these characters to a variable of type Str255.

In the case of the third (password) edit text control, GetDialogItemAsControl gets a handle to the control. GetControlDataSize is then called with the kControlEditTextTextTag tag to get the number of characters and GetControlData is called with the kControlEditTextPasswordTag to get the clear password text from the control.

DoExtractDateTime

DoExtractDateTime gets the settings in the three clock controls into a long date structure, extracts the content of the relevant fields of that structure, and draws that content in the image well area.

For each clock, GetControlData is called with the kControlClockLongDateTag tag to get the information into a long date structure. The rest of the code simply converts the values in the relevant fields of this structure to strings, which are concatenated with other identifying strings prior to being drawn in the image well.

NumericFilter

NumericFilter is the key filter function attached to the second edit text control.

The character code passed to this function by the control is first examined to determine whether it represents the numeric characters between 0 and 9 inclusive. If so, the function returns kControlKeyFilterPassKey, meaning that the character is to be accepted. If a return is not effected at that point, the character code is further examined to determine whether it represents one of the arrow keys or the "dash" character. If so, kControlKeyFilterPassKey is returned. If not, the system alert sound is played and kControlKeyFilterBlockKey is returned, meaning that the character is to be rejected.

EditTextValidator

EditTextValidator is the edit text validation function attached to the first edit text control.

The call to GetControlData gets the text to be examined from the control into the variable oldText. The next block sets the length byte in oldText to the actual number of characters in the text, limited to 255 characters. The for loop then examines each character and, if it is an alphabetic character, the space character, or the period character, inserts it into the newText variable and increments a variable which is eventually used to set the length byte of newText. When the if loop exits, the length byte is set and SetControlData is called to put the replacement text into the control. Finally, Draw1Control redraws the control.

GroupArrowsProgress.p

GroupArrowsProgress.p creates a movable modal dialog in which is demonstrated a checkbox group box, a pop-up group box, little arrows, a disclosure triangle, asynchronous arrows, and an indeterminate progress bar.

The checkbox group box contains a static text item and three radio buttons. The radio buttons are embedded in a radio group. The pop-up group box contains two user panes. Embedded in each user pane are a checkbox and a radio group. Embedded in each radio group are two radio buttons.

The dialog item list used by the dialog created by GroupArrowsProgress.p is shown in the following, in which the indentation represents the embedding hierarchy:

1. Push button primitive.
2. Group box control (kControlGroupBoxCheckboxProc variant).
    3. Radio group control.
        4. Radio button primitive.
        5. Radio button primitive.
        6. Radio button primitive.
    7. Checkbox primitive.
8. Group box control(kControlGroupBoxPopupButtonProc variant).
    9. User pane control with kControlSupportsEmbedding feature bit set (initial value set to 2).
        10. Radio group control
            11. Radio button primitive.
            12. Radio button primitive.
        13. Checkbox primitive.
        14. Static text primitive.
    15. User pane control with kControlSupportsEmbedding feature bit set (initial value set to 2).
        16. Radio group control
            17. Radio button primitive.
            18. Radio button primitive.
        19. Checkbox primitive.
        20. Static text primitive.
21. Group box control (kControlGroupBoxSecondaryTextTitleProc variant).
    22. Static text primitive.
23. Group box control (kControlGroupBoxSecondaryTextTitleProc variant).
    24. Static text primitive.
    25. Placard control.
        26. Static text primitive.
    27. Little arrows control.
28. Extract push button primitive.
29. Image well
30. Separator line
31. Disclosure triangle (kControlTriangleProc variant).
32. Static text primitive.

A second dialog item list resource, containing an asynchronous arrows control and a progress bar control, is appended to the dialog when the disclosure triangle is clicked.

DoTabEditClock

DoTabEditClock creates the dialog, calls ModalDialog to handle events until the user hits the OK push button, and then disposes of the dialog.

At the first two lines, if the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window is open, it is explicitly deactivated.

The call to GetNewDialog creates the dialog and SetDialogDefaultItem establishes the OK push button as the default push button.

The next two lines create routine descriptors for the application-defined event filter function and an application-defined action function for the little arrows control. (The event filter function is in the source code file Callbacks.p.)

The next block sets the initial value of the checkbox group box to 1 (checked).

The little arrows are used to set a (simulated) cache size, which is displayed in a static text item overlayed on a placard item. The little arrows initial value is set in the 'CNTL' resource to 96. The first three lines of the next block extract this value, convert it to a string and append the character "K". SetControlData is then called with the kControlStaticTextTextTag to set this string in the static text item.

The second user pane in the pop-up group box is then hidden before the call to ShowWindow, following which the ModalDialog loop is entered. Note that the UPP associated with the application-defined event filter function is passed in the filterProc parameter of ModalDialog.

If ModalDialog reports that the item hit was the checkbox group box's checkbox or the pop-up group box's pop-up menu, application-defined routines are called to further handle the hit.

if the item hit was one of the checkboxes within the pop-up group box, the control's control value is set so that the control is checked if it was unchecked, or unchecked if it was checked.

If the item hit was the disclosure triangle, an application-defined routine is called to further handle the hit.

If the item hit was the Extract push button, Draw1Control is called to redraw the image well, thus erasing any previous text in the well, and an application-defined routine is called to ectract the control settings and draw them in the image well area.

Note that hits on the radio buttons will be handled automatically by the radio group controls in which they are embedded. The radio group control will set the new control values according to which radio button is hit, and will change the checked/unchecked appearance of the radio buttons to match.

Note also that mouse-down events in the little arrows are detected and handled in the event filter function. (See the source code file Callbacks.c.)

When the OK button is hit, the loop exits, DisposeDialog closes the dialog and disposes of the associated memory, and the two previously created routine descriptors are disposed of. In addition, the global variable which flags that the dialog was open is set to false.

DoCheckBoxGroupBox

DoCheckBoxGroupBox further handles a hit in the checkbox group box's checkbox.

The first block flips the control's value; that is, if the control's value was 1 (checked), it is set to 0 (unchecked), and vice versa.

If the new value of the control is 0 (unchecked), the radio group and checkbox within the group box are deactivated. Since the radio buttons are embedded in the radio group, they will also be deactivated when the radio group is deactivated.

If the new value of the control is 1 (checked), the radio group and checkbox within the group box are activated. Since the radio buttons are embedded in the radio group, they will also be activated when the radio group is activated.

DoPopupGroupBox

DoPopupGroupBox further handles a hit in the pop-up group box's pop-up menu button.

The first two lines get the control's value, which represents the menu item chosen. Depending on the item chosen, the appropriate user panes are hidden or shown, thus hiding or showing all controls embedded in each user pane.

DoAsynchronousAndProgress

DoAsynchronousAndProgress further handles a hit in the disclosure triangle.

The first block flips the control's value; that is, if the control's value was 1 (expanded), it is set to 0 (collapsed), and vice versa.

If the new control value is 1 (expanded), GetResource is called to load the item list resource containing the asynchronous arrows and progress bar items. AppendDITL is then called to append these items to the end of the dialog's item list. The constant passed in the third parameter of this call causes the dialog box to be expanded downwards to accommodate the new items. SetControlData is then called with the kControlProgressBarIndeterminateTag tag to make the progress bar an indeterminate progress bar.

Not previously mentioned is the fact that the appended 'DITL' resource contains a static text item, with not text, the function of which is simply to expand the dialog further downwards to accommodate a move of the OK push button. The call to MoveControl moves the OK push button to a position below the asynchronous arrows and progress bar.

Finally, SetControlData is called with the kControlStaticTextTextTag tag to change the text in the static text field adjacent to the disclosure triangle. Draw1Control re-draws the control to display the changed text.

If the new disclosure triangle control value is 0 (collapsed), the OK button is moved back to its previous location, SizeWindow is called to resize the window to its previous size, SetControlData is called with the kControlStaticTextTextTag tag to change the text in the static text field adjacent to the disclosure triangle, and Draw1Control re-draws the control to display the changed text.

DoExtractCurrentStatus

DoExtractCurrentStatus is called when the Extract push button is hit. It simply extracts the values of the various controls, builds up strings based on those values, and draws those strings in the image well area.

Note in the second last block that the value of the little arrows control represents the current (simulated) cache size. The little arrows control value is set within the action function arrowsActionFunction in the source code file Callbacks.p.

Sliders.p

Sliders.p creates a movable modal dialog box in which is demonstrated various slider control variants and two simple user pane functions.

DoSliderUserPane

DoSliderUserPane creates the dialog, calls ModalDialog to handle events until the user hits the OK push button, and then disposes of the dialog.

At the first two lines, if the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window is open, it is explicitly deactivated.

The call to GetNewDialog creates the dialog and SetDialogDefaultItem establishes the OK push button as the default push button.

The next three lines create routine descriptors for the application-defined event filter function and two application-defined action functions for the live-feedback slider variants, one for the two at top right of the dialog and one for the two horizontal sliders. (The event filter function and the action functions are in the source code file Callbacks.p.)

The next two blocks creates routine descriptors for two user pane functions (one a draw function and one an activate function), and attach these functions to a user pane located immediately to the right of the two horizontal sliders.

The next block assign handles to the first four sliders to global variables and call an application-defined routine to draw the respective control values in static text items located immediately below the sliders.

The following four lines assign handles to the horizontal sliders to global variables and assign a value based on the current slider control values to the red and blue fields of two global variables of type RGBColor.

The call to ShowWindow makes the dialog visible before the ModalDialog loop is entered. This loop executes until the OK button is hit, at which time DisposeDialog is called to remove the dialog and dispose of its associated memory, the previously created routine descriptors are disposed of, and the global variable which flags that the Sliders dialog is open is assigned false.

DoDrawSliderValues

DoDrawSliderValue is called by DoSliderUserPane to draw the initial control values of the four top sliders in the static text fields immediately below the sliders.

The first line gets the control's value and converts it to a string. The next block determines, on the basis of the received control handle, which is the target static text item. SetControlData is then called with the kControlStaticTextTextTag tag to set the text, and Draw1Control re-draws the static text control.

As will be seen, this function is also called from within the event filter function when a mouse-down has been detected within the two non-live-scrolling sliders.

UserPaneDrawFunction

UserPaneDrawFunction will be called whenever an update event is received. This will occur when the dialog is opened and, subsequently, when an overlaying window or help balloon is removed from the area occupied by the user pane (to the immediate right of the horizontal sliders).

The two calls to DrawThemeGenericWell draw two Appearance image well primitives to the right of the horizontal sliders. The global variable gDrawActivated determines whether the primitive is drawn with an activated or deactivated appearance.

If the global variable gDrawActivated indicates that the interior of the well should be drawn, the foreground colour for each interior is set to that stored in global variables and PaintRect is called to paint the interior in that colour. (As will be seen, the value stored in these global variables is changed by the action function called while the mouse button remains down in the slider's indicators.)

If gDrawActivated indicates that the interior of the well should be displayed with a deactivated appearance, the interior is left as drawn by DrawThemeGenericWell, that is, white.

UserPaneActivateFunction

The kControlWantsActivate feature bit is set in the user pane on creation, that is, the control's minimum value in the 'CNTL' resource is set to 16. This ensures that this function will be called when the dialog receives activate events.

Depending on whether the dialog is about to be activated or deactivated, the global variable gDrawActivated is set to true or false. As has been seen, this global variable controls the way in which the image well primitive and the interior of the image well is drawn.

TextBox.c

DoAboutBox is called when the user chooses About Controls3... from the Apple menu. It opens a movable modal dialog containing a standard and scrolling text box controls.

Callbacks.p

Callbacks.p contains the event filter function used by all movable modal dialogs and three action functions, one for the little arrows, one for the live-feedback sliders at top right of the dialog, and one for the horizontal live-feedback sliders.

EventFilter

The event filter function EventFilter is identical to the event filter function introduced at Chapter 8 — Dialogs and Alerts except that it intercepts mouse-down events in the little arrows and sliders and informs ModalDialog that it has handled those events.

At the second and third if statements, if the Group Boxes, Arrows, and Progress Bar dialog is currently open, and if the event is a mouse-down event, the location of the mouse-down is extracted from the where field of the event structure and converted to the local coordinates required by the following call to FindControl. If FindControl determines that there is an enabled control under the mouse cursor, and if that control is the little arrows control, TrackControl is called to handle all user interaction while the mouse button remains down. While the mouse button remains down, TrackControl continually calls the action function whose routine descriptor is pointed to by the UPP passed in the third parameter. When the mouse button is released, the function exits with true being returned to ModalDialog.

if, on the other hand, the Sliders and User Pane Functions dialog is open, and if the event is a mouse-down event, and if FindControl reports an enabled control, and if that control is one of the two non-live-feedback sliders, TrackControl is called to handle all user interaction while the mouse button remains down. When the mouse button is released, the application-defined routine DoDrawSliderValues is called to draw the new control value in the static text item below the slider. Once again, the function then exits with true being returned to ModalDialog.

If the mouse-down was in one of the two live-feedback sliders at the top right of the dialog, TrackControl is called with a UPP to the first slider action function passed in the third parameter.

If the mouse-down was in one of the two horizontal live-feedback sliders, TrackControl is called with a UPP to the second slider action function passed in the third parameter.

ArrowsActionFunction

ArrowsActionFunction is the action function for the little arrows. It is called continually by TrackControl while the mouse button remains down.

if the mouse cursor is still within the control, GetControlValue is called to get the current control value. The function then switches according to the part code. If the cursor is in the top arrow, the control's value is increased by 32 unless this would cause the value to exceed the control's maximum value (set in the 'CNTL' resource to 7680). If the cursor is in the bottom arrow, the control's value is decreased by 32 unless this would cause the value to be lower that the control's minimum value (set in the 'CNTL' resource to 96).

If the control's value was increased or decreased within the switch, SetControlValue is called to set the new control value, and the final block sets the new text for the Cache size static text item and re-draws the item.

SliderActionFunction1

SliderActionFunction1 is the action function for the live-feedback sliders at the top right of the dialog. It is called continually by TrackControl while the mouse button remains down.

GetControlValue gets the control's current value and NumToString converts it to a string for display. The next line gets the pointer to the control's owning window required by the call to GetDialogItemAsControl. The if block determines which of the two sliders the mouse is down in. The last two lines set the text in the appropriate static text control and redraw the control.

SliderActionFunction2

SliderActionFunction3 is the action function for the horizontal live-feedback sliders. It is called continually by TrackControl while the mouse button remains down.

GetControlValue gets the control's current value. If the slider is the top horizontal slider, the red field of the RGBColor global variable gRedColor is assigned a new value based on the control's current value, and the local Rect variable is assigned the coordinates of the image well at the right of the slider. If the slider is the bottom horizontal slider, the blue

field of the RGBColor global variable gBlueColor is assigned a new value based on the control's current value, and the local Rect variable is assigned the coordinates of the image well at the right of the slider.

Finally, PaintRect is called to paint the interior area of the relevant image well in the new colour.