14

MORE ON CONTROLS

Demonstration Program: Controls3

Introduction

Chapter 7 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 (text title variant) and user panes. This chapter revisits group boxes and user panes and addresses the many remaining controls.

The controls addressed in this chapter are as follows:

Bevel button	Image well	Tab	Edit text	Slider
Group boxes	Clock	Progress and relevance bar	Little arrows	Disclosure triangle
Picture	Icon	Window header	Placard	Static text
Separator line	Pop-up arrow	Radio group	Chasing arrows	User pane
Scrolling text box	Disclosure button	Edit Unicode text	Round button	

The data browser control is not addressed in this book. The one remaining control (the list box) is addressed at Chapter 22.

The Progress bar 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 demonstration program associated with Chapter 25.

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 were **embedding** and **latency**. Embedding hierarchies have implications for drawing order and hit testing.
- Another feature introduced with Mac OS 8 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, if the `kDialogFlagsUseControlHierarchy` feature bit is set in a dialog's `'dlgx'` resource, the Dialog Manager creates a root control in the dialog 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 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

Bevel Buttons

A bevel button is a rectangular control with a bevelled edge that can include text, an icon, a picture, or a combination of text and an icon or picture. Bevel buttons can be made to behave in a number of different ways: 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.

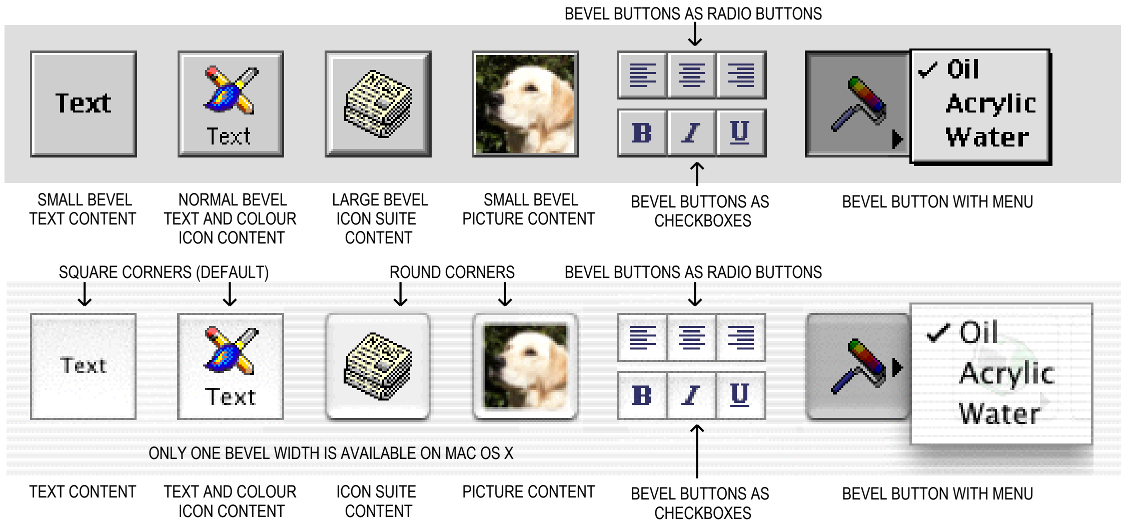
BEVEL BUTTONS AS RADIO BUTTONS
Text
Text
✓ Oil
Acrylic
Water
SMALL BEVEL TEXT CONTENT
NORMAL BEVEL TEXT AND COLOUR ICON CONTENT
LARGE BEVEL ICON SUITE CONTENT
SMALL BEVEL PICTURE CONTENT
BEVEL BUTTONS AS CHECKBOXES
BEVEL BUTTON WITH MENU
SQUARE CORNERS (DEFAULT)
ROUND CORNERS
BEVEL BUTTONS AS RADIO BUTTONS
Text
Text
✓ Oil
Acrylic
Water
ONLY ONE BEVEL WIDTH IS AVAILABLE ON MAC OS X
TEXT CONTENT
TEXT AND COLOUR ICON CONTENT
ICON SUITE CONTENT
PICTURE CONTENT
BEVEL BUTTONS AS CHECKBOXES
BEVEL BUTTON WITH MENU

FIG 1 - TYPICAL BEVEL BUTTONS

Note that, on Mac OS X, bevel button have square corners by default but can be made rounded using the function `SetControlData` with the `kControlBevelButtonKindTag` tag (see Other Control Data Tag Constants, below).

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`

Note that, while three bevel sizes are available on Mac OS 8/9, bevel button will always be displayed with one bevel size on Mac OS X regardless of which of the first three variants is used.

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:

Constant	Value	Description
`kControlBehaviorPushbutton`	`0`	Button pops up after being clicked. This constant is used when push button behaviour is required.
`kControlBehaviorToggles`	`0x0100`	Button toggles state automatically when clicked. This constant is used when checkbox or radio button behaviour is required.
`kControlBehaviorSticky`	`0x0200`	When clicked, button stays down until application sets control’s value to 0. This behaviour is useful in tool palettes.
`kControlBehaviorOffsetContents`	`0x8000`	Button contents are offset (one pixel down and to the right) when button is clicked. (Some users consider that this behaviour gives a bevel button a more realistic "feel".)

Bevel Button Behaviour Constants (Menus)

The following bevel button behaviour constants (menus) apply to the high byte of a bevel button's minimum value:

Constant	Value	Description
`kControlBehaviorSingleValueMenu`	`0`	Menu contains commands, not choices, and should not be marked with a checkmark. Overrides the `kControlBehaviorMultiValueMenu` bit.
`kControlBehaviorMultiValueMenu`	`0x4000`	Menus are multi-valued. The user can toggle entries in the button's menu and have multiple items checked. The button does not maintain the menu value as it normally would. When this bit is set, the menu value accessed using `GetControlData` with the `kControlBevelButtonMenuValueTag` will return the value of the last menu item chosen.

Button and Image Well Content Type Constants

The following **button and image well content type constants** apply to the low byte of a bevel button's minimum value:

Constant	Value	Description
`kControlContentTextOnly`	`0`	Content type is text only.
`kControlContentIconSuiteRes`	`1`	Content type is an icon suite resource ID.
`kControlContentCIconRes`	`2`	Content type is a colour icon resource ID.
`kControlContentPictRes`	`3`	Content type is a picture resource ID.
`kControlContentIconSuiteHandle`	`129`	Content type is an icon suite handle.
`kControlContentCIconHandle`	`130`	Content type is a colour icon handle.
`kControlContentPictHandle`	`131`	Content type is a picture handle.
`kControlContentIconRef`	`132`	Content type is an icon reference.

You can also use these constants in the `contentType` field of the **button content info 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), handle (for handle-based content), or reference (for reference-based content) of a colour icon, icon suite, picture, or icon reference in a bevel button.[1]

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 icons, pictures, and text in a bevel button, and you can specify the placement of text in relation to an icon or picture. By calling `GetControlData` with these constants you can also ascertain alignment and placement.

[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.

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`	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. (See Bevel Button Text Alignment Constants, above.) **Data type returned or set:** `ControlButtonTextAlignment`
`kControlBevelButtonGraphicAlignTag`	Gets or sets the alignment of graphics in 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 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 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 is offset from the button's edges. Ignored when the graphic is specified to be centred on the button. **Data type returned or set:** Point

The Button Content Info Structure

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

```
struct ControlButtonContentInfo
{
  ControlContentType  contentType;
  union {
    SInt16      resID;
    CIconHandle cIconHandle;
    Handle      iconSuite;
    IconRef     iconRef;
    PicHandle   picture;
    Handle      ICONHandle;
  } u;
};
typedef struct ControlButtonContentInfo ControlButtonContentInfo;
typedef ControlButtonContentInfo *ControlButtonContentInfoPtr;
typedef ControlButtonContentInfo ControlImageContentInfo;
typedef ControlButtonContentInfo *ControlImageContentInfoPtr;
```

Field Descriptions

contentType	Specifies the content type (see Bevel Button and Image Well Content Type Constants, above.) and determines which of the following fields are used.
resID	If the content type specified in the contentType field is kControlContentIconSuiteRes, kControlContentCIconRes, or kControlContentPictRes, this field should contain 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 should contain a handle to a colour icon.
iconSuite	If the content type specified in the contentType field is kControlContentIconSuiteHandle, this field should contain a handle to an icon suite.
iconRef	If the content type specified in the contentType field is kControlContentIconRef, this field should contain an icon reference.
picture	If the content type specified in the contentType field is kControlContentPictHandle, this field should contain 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 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. See the Icons section at Chapter 13.) **Data type returned or set:** IconTransformType

`kControlBevelButtonMenuValueTag`	Gets the menu value. (See "Bevel Button Menu Constants, above.) **Data type returned:** `SInt16`
`kControlBevelButtonMenuRefTag`	Gets the menu reference. **Data type returned:** `MenuRef`
`kControlBevelButtonLastMenuTag`	Gets the menu ID of the last menu selected in the submenu or main menu. **Data type returned:** `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.
`kControlBevelButtonContentTag`	Sets bevel button content. **Data type reset:** `ButtonContentInfo`
`kControlBevelButtonOwnedMenuRefTag`	Sets the menu reference. (The control will dispose.) **Data type set:** `MenuRef`
`kControlBevelButtonKindTag`	Sets the button kind. **Data type returned or set:** `ThemeButtonKind (kThemeSmallBevelButton, kThemeMediumBevelButton, kThemeLargeBevelButton, kThemeRoundedBevelButton).`

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.

Helper Functions

The following helper functions may be used in lieu of `SetControlData` and `GetControlData` with the relevant control data tags:

```
GetBevelButtonMenuValue
SetBevelButtonMenuValue
GetBevelButtonMenuHandle
GetBevelButtonContentInfo
SetBevelButtonContentInfo
SetBevelButtonTransform
SetBevelButtonGraphicAlignment
SetBevelButtonTextAlignment
SetBevelButtonTextPlacement
```

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. 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.

Programmatic Creation

Bevel button controls may be created programmatically using the function `CreateBevelButtonControl`:

```
OSStatus  CreateBevelButtonControl(WindowRef window,const Rect *boundsRect,
                                   CFStringRef title,ControlBevelThickness thickness,
                                   ControlBevelButtonBehavior behavior,
                                   ControlButtonContentInfoPtr info,SInt16 menuID,
                                   ControlBevelButtonMenuBehavior menuBehavior,
                                   ControlBevelButtonMenuPlacement menuPlacement,
                                   ControlRef *outControl);
```

Relevant constants are:

Parameter	Constants

`thickness`	`kControlBevelButtonSmallBevel`
	`kControlBevelButtonNormalBevel`
	`kControlBevelButtonLargeBevel`

`behavior`	See Bevel Button Behaviour Constants, above.
`info.contentType`	See Button and Image Well Content Type Constants, above.
`menuBehavior`	See Bevel Button Behaviour Constants (Menus), above.
`menuPlacement`	`kControlBevelButtonMenuOnBottom` `kControlBevelButtonMenuOnRight`

Image Wells

Image wells can be used to display icons or pictures. They 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.

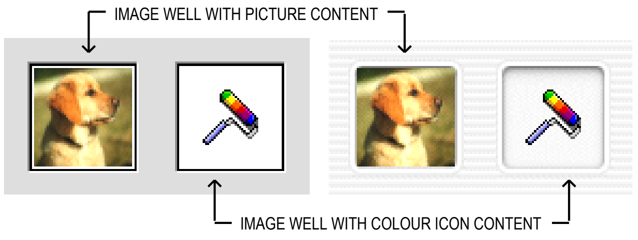
IMAGE WELL WITH PICTURE CONTENT
IMAGE WELL WITH COLOUR ICON CONTENT

FIG 2 - TYPICAL IMAGE WELLS

Variants and Control Definition IDs

The image well CDEF resource ID is 11. The one available variant and its control definition IDs is as follows:

Variant	Var Code	Control Definition ID
Image well.	`0`	`176    kControlImageWellProc`

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 Button Content Info structure

The button and image well content type constants (see Bevel Buttons, above) apply to an image well's minimum value.

You can also use these constants in the `contentType` field of the button content info 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. (See The Button Content Info Structure, above.) **Data type returned or set:** `ControlButtonContentInfo` structure
`kControlImageWellTransformTag`	Gets or sets a transform that is added to the standard transform. (See the Icons section at Chapter 13.) **Data type returned or set:** `IconTransformType`
`kControlImageWellIsDragDestinationTag`	**Data type returned or set:** `Boolean`

Helper Functions

The following helper functions may be used in lieu of `SetControlData` and `GetControlData` with the relevant control data tag constants:

```
GetImageWellContentInfo
SetImageWellContentInfo
SetImageWellTransform
```

Control Part Codes

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

Programmatic Creation

Image well controls may be created programmatically using the function `CreateImageWellControl`:

```
OSStatus  CreateImageWellControl(WindowRef window,const Rect *boundsRect,
                                const ControlButtonContentInfo *info,
                                ControlRef *outControl);
```

Relevant constants are:

Parameter	Constants
`info.contentType`	See Button and Image Well Content Type Constants, above.

The tab control is an embedding control which provides a means of presenting information on multiple "pages". The user selects the desired "page" by clicking the appropriate tab.

A typical tab control is shown at Fig 3.

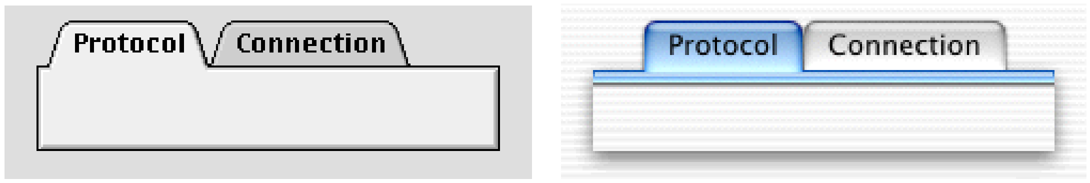

FIG 3 - TAB CONTROL

The content area of a tab is known as a **pane**. Controls which are embedded within an individual pane should only affect the settings displayed in that 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.

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

Variants and Control Definition IDs

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

Variant	Var Code	Control Definition ID	
Large tab control, tabs at top.	0	128	`kControlTabLargeNorthProc`
Small tab control, tabs at top.	1	129	`kControlTabSmallNorthProc`
Large tab control, tabs at bottom.	2	130	`kControlTabLargeSouthProc`
Small tab control, tabs at bottom.	3	131	`kControlTabSmallSouthProc`
Large tab control, tabs at right.	4	132	`kControlTabLargeEastProc`
Small tab control, tabs at right.	5	133	`kControlTabSmallEastProc`
Large tab control, tabs at left.	6	134	`kControlTabLargeWestProc`
Small tab control, tabs at left.	7	135	`kControlTabSmallWestProc`

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. **Data type returned:** `Rect`
`kControlTabEnabledFlagTag`	Enables or disables a single tab. **Data type returned or set:** `Boolean`; if `true`, enabled; if `false`, disabled.
`kControlTabFontStyleTag`	Gets or sets the font and font style. **Data type returned or set:** `ControlFontStyleRec`
`kControlTabInfoTag`	Gets or sets information for a tab in a tab control. (See The Tab Information Structure, below.) **Data type returned or set:** `ControlTabInfoRec`

Helper Functions

The following helper functions may be used in lieu of `SetControlData` and `GetControlData` with the relevant control data tags:

```
GetTabContentRect
SetTabEnabled
```

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 should be one of the following:

```
struct ControlTabInfoRec
{
  SInt16 version;      // Set to kControlTabInfoVersionZero.
  SInt16 iconSuiteID;  // Set to 0 for no icon.
  Str255 name;         // Title for tab label.
};
typedef struct ControlTabInfoRec ControlTabInfoRec;

struct ControlTabInfoRecV1
{
  SInt16      version;     // Set to kControlTabInfoVersionOne.
  SInt16      iconSuiteID; // Set to 0 for no icon.
  CFStringRef name;        // Title for tab label.  (Should always be released)
};
typedef struct ControlTabInfoRecV1 ControlTabInfoRecV1;
```

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. Fig 4 shows the structure of a compiled `'tab#'` resource.

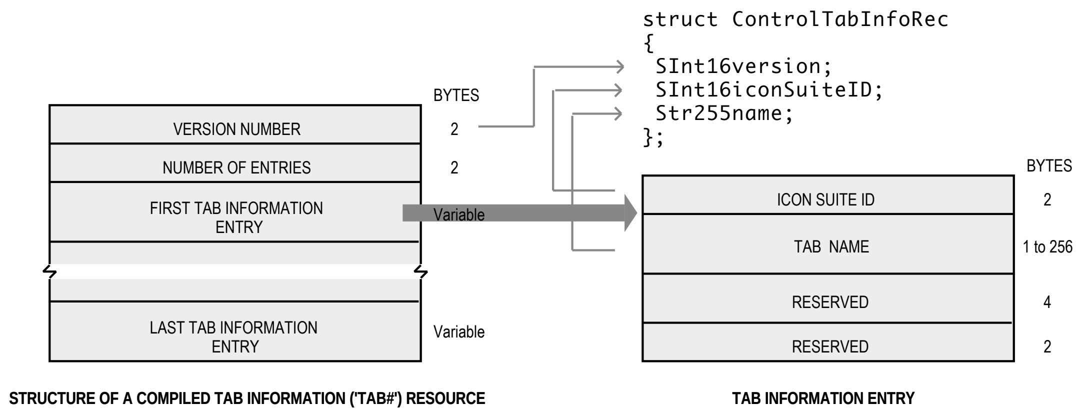

FIG 4 - STRUCTURE OF A COMPILED TAB INFORMATION ('tab#') RESOURCE

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

Field	Description
VERSION NUMBER	Version of the resource.
NUMBER OF ENTRIES	The number of tab information entries in the resource.
ICON SUITE ID	Icon suite resource ID.
TAB NAME	The tab name.

Programmatic Creation

Tab controls may be created programmatically using the function `CreateTabsControl`:

```
OSStatus  CreateTabsControl(WindowRef window,const Rect *boundsRect,ControlTabSize size,
                            ControlTabDirection direction,UInt16 numTabs,
                            const ControlTabEntry *tabArray,
                            ControlRef *outControl);
```

Relevant constants are:

Parameter	***Constants***
`size`	`kControlTabSizeLarge` `kControlTabSizeSmall`

`direction`	`kControlTabDirectionNorth`
	`kControlTabDirectionSouth`
	`kControlTabDirectionEast`
	`kControlTabDirectionWest`

The `tabArray` parameter of takes a pointer to an array of **tab entry structures**:

```
struct ControlTabEntry
{
  ControlButtonContentInfo *icon;
  CFStringRef              name;
  Boolean                  enabled;
};
typedef struct ControlTabEntry ControlTabEntry;
```

Edit Text

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

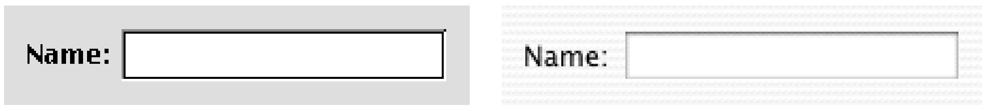

FIG 5 - EDIT TEXT CONTROL

Edit text controls can have a key filter function attached to filter key strokes or modify them on return.

Variants and Control Definition IDs

The edit text control 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.	`0`	272	`kControlEditTextProc`
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.

Main Control Data Tag Constants

Control Data Tag Constant	Meaning and Data Type Returned or Set
`kControlEditTextTextTag`	Gets or sets text. **Data type returned or set:** character buffer
`kControlEditTextTEHandleTag`	Gets a handle to the text edit structure. Data type returned or set: `TEHandle`
`kControlEditTextSelectionTag`	Gets or sets the selection. **Data type returned or set:** `ControlEditTextSelectionRec` structure. (See The Edit Text Selection Structure, below.)
`kControlEditTextPasswordTag`	Gets clear password text, 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`
`kControlEditTextCFStringTag`	Gets or sets text. (Mac OS X only.) **Data type returned or set:** `CFStringRef`

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`:

```
struct ControlEditTextSelectionRec
{
  SInt16 selStart;
  SInt16 selEnd;
};
typedef struct ControlEditTextSelectionRec ControlEditTextSelectionRec;
typedef ControlEditTextSelectionRec *ControlEditTextSelectionPtr;
```

Field Descriptions

`selStart`	Beginning of the edit text selection.
`selEnd`	End of the edit text selection.

Programmatic Creation

Edit text controls may be created programmatically using the function `CreateEditTextControl`:

```
OSStatus  CreateEditTextControl(WindowRef window,const Rect *boundsRect,CFStringRef text,
                                Boolean isPassword,Boolean useInlineInput,
                                const ControlFontStyleRec *style,ControlRef *outControl);
```

Sliders

A slider control comprises a **slider bar** and an **indicator**. The user can drag the indicator to set a new value within the range represented by the slider bar.

Sliders can be oriented horizontally or vertically, and the indicator can point in any direction. The indicator can also be nondirectional if required. Typical sliders are shown at Fig 6.

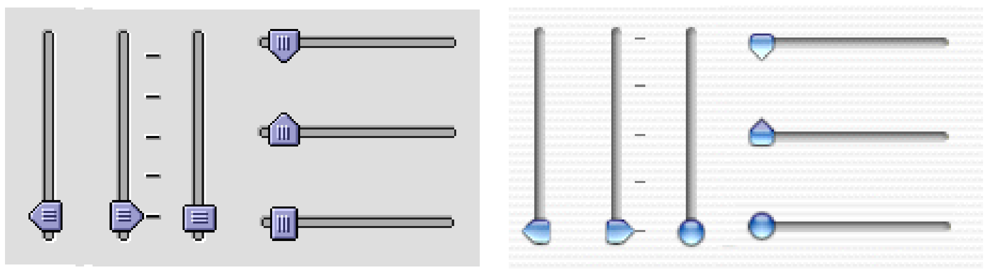

FIG 6 - SLIDERS

A slider can, optionally, display tick marks, and you can specify the number of tick marks required. If you specify tick marks, you should ensure that they are labelled.

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.

Programmatic Creation

Slider controls may be created programmatically using the function `CreateSliderControl`:

```
OSStatus  CreateSliderControl(WindowRef window,const Rect *boundsRect,SInt32 value,
                              SInt32 minimum,SInt32 maximum,
                              ControlSliderOrientation orientation,UInt16 numTickMarks,
                              Boolean liveTracking,ControlActionUPP liveTrackingProc,
                              ControlRef *outControl);
```

Relevant constants are:

Parameter	Constants
orientation	kControlSliderPointsDownOrRight kControlSliderPointsUpOrLeft kControlSliderDoesNotPoint

Group Boxes — Embedding Control

Group boxes are embedding controls used to group a number of related items. They may be either **primary** or **secondary**.

A 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 grouping subsidiary information.

Typical group boxes are shown at Fig 7.

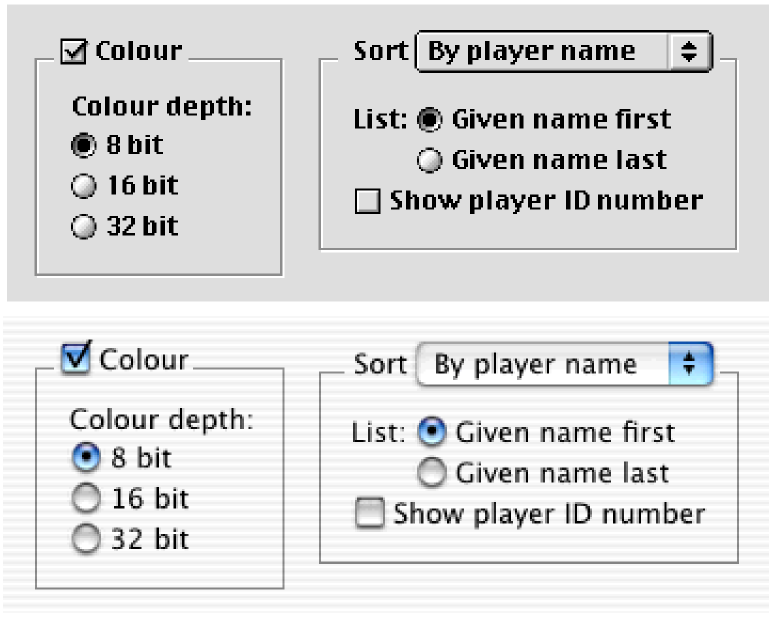

FIG 7 - TYPICAL GROUP BOXES

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
`kControlGroupBoxMenuRefTag`	Gets the menu reference of a group box. **Data type returned or set:** `MenuRef`
`kControlGroupBoxFontStyleTag`	Gets or sets the font style. **Data type returned or set:** `ControlFontStyleRec`

`kControlGroupBoxTitleRectTag`	Gets the rectangle containing the group box title (and any associated control, such as a checkbox). **Data type returned:** `Rect`

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.

Programmatic Creation

Group box controls may be created programmatically using the functions `CreateGroupBoxControl`, `CreateCheckGroupBoxControl`, and `CreatePopupGroupBoxControl`:

```
OSStatus  CreateGroupBoxControl(WindowRef window,const Rect *boundsRect,
                                CFStringRef title,Boolean primary,
                                ControlRef *outControl);

OSStatus  CreateCheckGroupBoxControl(WindowRef window,const Rect *boundsRect,
                                     CFStringRef title,SInt32 initialValue,
                                     Boolean primary,Boolean autoToggle,
                                     ControlRef *outControl);

OSStatus  CreatePopupGroupBoxControl(WindowRef window,const Rect *boundsRect,
                                     CFStringRef title,Boolean primary,SInt16 menuID,
                                     Boolean variableWidth,SInt16 titleWidth,
                                     SInt16 titleJustification,Style titleStyle,
                                     ControlRef *outControl);
```

Clock

Clock controls are a combination of an edit text control and little arrows (see below) which display date or time. The displayed date and time may be changed using the little arrows or by typing in the edit text control. Fig 8 shows a clock control displaying a date.

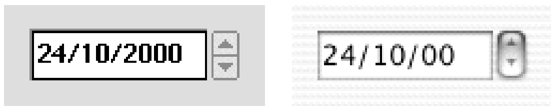

FIG 8 - CLOCK CONTROL DISPLAYING DATE

The clock control supports keyboard navigation and keyboard focus. If the control is made inactive, the user will be unable to change the displayed date or time values; however the correct date and time will continue to be displayed.

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`
`kControlClockAnimatingTag`	Starts and stops clock animation. Use only if a call to `GetControlFeatures` reveals that the control's `kControlIdlesWithTimer` flag is set. **Data type set:** `Boolean`

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.

Programmatic Creation

Clock controls may be created programmatically using the function `CreateClockControl`:

```
OSStatus  CreateClockControl(WindowRef window,const Rect *boundsRect,
                             ControlClockType clockType,ControlClockFlags clockFlags,
                             ControlRef *outControl);
```

Relevant constants are:

Parameter	Constants

`clockType`	`kControlClockTypeHourMinute`
	`kControlClockTypeHourMinuteSecond`
	`kControlClockTypeMonthDayYear`
	`kControlClockTypeMonthYear`

`clockFlags`	See Clock Value Flag Constants, above.

Progress and Relevance Bars

Progress bars are used to indicate capacity or the current status of a lengthy operation. Two types of Progress bars can be used: an **indeterminate progress bar**, which shows that an operation is occurring but does not indicate its current status; a **determinate progress bar**, which shows how much of the operation has been completed. The two types are shown at Fig 9.

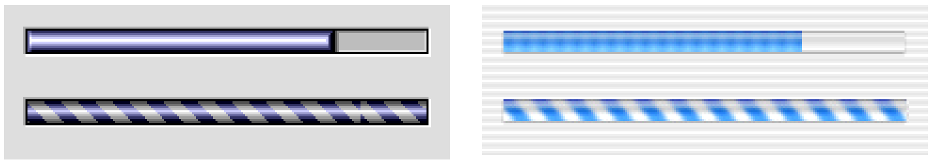

FIG 9 - PROGRESS INDICATORS

On Mac OS 8/9, the progress bar control is often used as a **relevance bar**. On Mac OS X, a separate relevance bar control, which can only be created programmatically, is available.

Variants and Control Definition IDs

The progress bar CDEF resource ID is 5. The two available variants and their control definition IDs are as follows:

Variant	Var Code	Control Definition ID	
Progress bar. To make the control determinate or indeterminate, set the `kControlProgressBarIndeterminateTag` constant. (See Control Data Tag Constant Relevant to Progress bars, below.)	0	80	`kControlProgressBarProc`
Relevance bar.	1	81	`kControlRelevanceBarProc`

Progress Bar Control Values

Control Value	Content
Initial	Appropriate value between –32768 and 32767.
Minimum	–32768 to 32767.
Maximum	–32768 to 32767.

Control Data Tag Constant

Control Data Tag Constant	Meaning and Data Type Returned or Set
`kControlProgressBarIndeterminateTag`	Gets or sets whether a progress bar is determinate or indeterminate. **Data type returned or set:** `Boolean`. If `true`, switches to an indeterminate progress bar. If `false`, switches to an determinate progress bar.
`kControlProgressBarAnimatingTag`	Starts and stops progress bar animation. Use only if a call to `GetControlFeatures` reveals that the control's `kControlIdlesWithTimer` flag is set. **Data type returned or set:** `Boolean`

Programmatic Creation

Progress bars may be created programmatically using the function `CreateProgessBarControl`:

```
OSStatus  CreateProgressBarControl(WindowRef window,const Rect *boundsRect,SInt32 value,
                                   SInt32 minimum,SInt32 maximum,Boolean indeterminate,
                                   ControlRef *outControl);
```

On Mac OS X only, relevance bars may be created programmatically using the function `CreateRelevanceBarControl`:

```
OSStatus  CreateRelevanceBarControl(WindowRef window,const Rect *boundsRect,SInt32 value,
                                    SInt32 minimum,SInt32 maximum,
                                    ControlRef *outControl);
```

Little Arrows

The little arrows control (see Fig 10), which consists of two opposing arrows, provide a means of increasing or decreasing values. They should always be accompanied by a label that identifies the content to which the control relates.

The displayed value is incremented or decremented by one unit of change when the user clicks the up or down arrow. If the user clicks an arrow and holds the mouse button down, the value increases or decreases until the user releases the button. The unit of change should depend on the content.

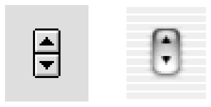

FIG 10 - LITTLE ARROWS

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 32767.
Minimum	–32768 and 32767.
Maximum	–32768 and 32767.

Control Part Codes

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

Programmatic Creation

Little arrows controls may be created programmatically using the function `CreateLittleArrowsControl`:

```
OSStatus  CreateLittleArrowsControl(WindowRef window,const Rect *boundsRect,SInt32 value,
                                    SInt32 minimum,SInt32 maximum,SInt32 increment,
                                    ControlRef *outControl);
```

Disclosure Triangles

Disclosure triangles (see Fig 11) are used to display and hide additional information in a window and also to reveal the contents of folders in list views. They have two possible values: 0 for collapsed and 1 for expanded. The first click on the control rotates the triangle downwards. The second click rotates the triangle back to the original orientation.

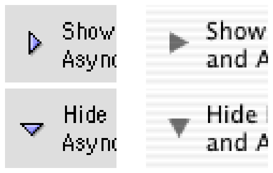

FIG 11 - DISCLOSURE TRIANGLES

Variants and Control Definition IDs

The disclosure triangle CDEF resource ID is 4. The four available variants and their 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`

Helper Function

The helper function `SetDisclosureTriangleLastValue` may be used in lieu of `SetControlData` with the `kControlTriangleLastValueTag` control data tag.

Control Part Codes

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

Programmatic Creation

Disclosure triangle controls may be created programmatically using the function `CreateDisclosureTriangleControl`:

```
OSStatus  CreateDisclosureTriangleControl(WindowRef window,const Rect *boundsRect,
                                          ControlDisclosureTriangleOrientation orientation,
                                          CFStringRef title,SInt32 initialValue,
                                          Boolean drawTitle,Boolean autoToggles,
                                          ControlRef *outControl);
```

Relevant constants are:

Parameter	***Constants***

`orientation`	`kControlDisclosureTrianglePointDefault`
	`kControlDisclosureTrianglePointRight`
	`kControlDisclosureTrianglePointLeft`

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 Data Tag Constant

Control Data Tag Constant	***Meaning and Data Type Returned or Set***
`kControlPictureHandleTag`	Sets picture. **Data type set:** `PicHandle`

Control Part Codes

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

Programmatic Creation

Picture controls may be created programmatically using the function `CreatePictureControl`:

```
OSStatus  CreatePictureControl(WindowRef window,const Rect *boundsRect,
                               const ControlButtonContentInfo *content,Boolean dontTrack,
                               ControlRef *outControl);
```

Icon

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

A non-tracking variant is available for use in dialogs 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.) **Data type returned or set:** `IconTransformType`
`kControlIconAlignmentTag`	Gets or sets an icon's position. (See Chapter 13.) **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.

Programmatic Creation

Icon controls may be created programmatically using the function `CreateIconControl`:

```
OSStatus  CreateIconControl(WindowRef window,const Rect *boundsRect,
                            const ControlButtonContentInfo *icon,Boolean dontTrack,
                            ControlRef *outControl);
```

Window Headers — Embedding Control

Window headers are rectangular embedding controls which should be located at the top of a window's content region and used to provide information about the window's contents.

The list view header variant is similar to the main variant, but removes the line that 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.

Programmatic Creation

Window header controls may be created programmatically using the function `CreateWindowHeaderControl`:

```
OSStatus  CreateWindowHeaderControl(WindowRef window,const Rect *boundsRect,
                                    Boolean isListHeader,ControlRef *outControl);
```

Placard — Embedding Control

Placards are rectangular embedding controls used to display information. They can also be used simply as background fill for a control area. Typically, placards are used as a small information panel placed at the bottom of a window.

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.

Programmatic Creation

Placard controls may be created programmatically using the function `CreatePlacardControl`:

```
OSStatus  CreatePlacardControl(WindowRef window,const Rect *boundsRect,
                               ControlRef *outControl);
```

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. **Data type returned or set:** character buffer
`kControlStaticTextTextHeightTag`	Gets the height of text. **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 end of a line. **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.
`kControlStaticTextCFStringTag`	Gets or sets a CFString reference. **Data type returned or set:** `CFStringRef`

Programmatic Creation

Static text controls may be created programmatically using the function `CreateStaticTextControl`:

```
OSStatus  CreateStaticTextControl(WindowRef window,const Rect *boundsRect,
                                  CFStringRef text,const ControlFontStyleRec *style,
                                  ControlRef * outControl);
```

Separator Lines

Separator lines are vertical or horizontal lines used to visually separate groups of controls. The orientation of the bounding rectangle determines the orientation of the 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.

Programmatic Creation

Separator line controls may be created programmatically using the function `CreateSeparatorControl`:

```
OSStatus  CreateSeparatorControl(WindowRef window,const Rect *boundsRect,
                                 ControlRef *outControl);
```

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, east-facing pop-up arrow.	0	192	kControlPopupArrowEastProc
Large, west-facing pop-up arrow.	1	193	kControlPopupArrowWestProc
Large, north-facing pop-up arrow.	2	194	kControlPopupArrowNorthProc
Large, south-facing pop-up arrow.	3	195	kControlPopupArrowSouthProc
Small, east-facing pop-up arrow.	4	196	kControlPopupArrowSmallEastProc
Small, west-facing pop-up arrow.	5	197	kControlPopupArrowSmallWestProc
Small, north-facing pop-up arrow.	6	198	kControlPopupArrowSmallNorthProc
Small, south-facing pop-up arrow.	7	199	kControlPopupArrowSmallSouthProc

Control Values

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

Programmatic Creation

Pop-up arrow controls may be created programmatically using the function `CreatePopupArrowControl`:

```
OSStatus  CreatePopupArrowControl(WindowRef window,const Rect *boundsRect,
                                  ControlPopupArrowOrientation orientation,
                                  ControlPopupArrowSize size,ControlRef *outControl);
```

Relevant constants are:

Parameter	Constants
orientation	kControlPopupArrowOrientationEast kControlPopupArrowOrientationWest kControlPopupArrowOrientationNorth kControlPopupArrowOrientationSouth
size	kControlPopupArrowSizeNormal kControlPopupArrowSizeSmall

Radio Groups

Radio groups are embedding 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 the 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.

Programmatic Creation

Radio group controls may be created programmatically using the function `CreateRadioGroupControl`:

```
OSStatus  CreateRadioGroupControl(WindowRef window,const Rect *boundsRect,
                                  ControlRef *outControl);
```

Chasing Arrows

Chasing arrows (see Fig 12) 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 containing a progress bar.

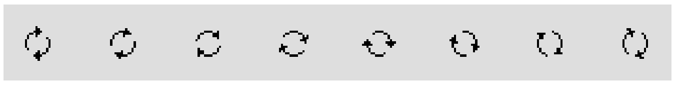

FIG 12 - FRAMES OF A CHASING ARROWS ANIMATION

Variant and Control Definition ID

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

Variant	Var Code	Control Definition ID
Chasing arrows.	0	112 `kControlChasingArrowsProc`

Control Values

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

Control Data Tag Constant

Control Data Tag Constant	Meaning and Data Type Returned or Set
`kControlChasingArrowsAnimatingTag`	Starts and stops chasing arrows animation. Use only if a call to `GetControlFeatures` reveals that the control's `kControlIdlesWithTimer` flag is set. **Data type returned or set:** `Boolean`.

Programmatic Creation

Asynchronous arrow controls may be created programmatically using the function `CreateChasingArrowsControl`:

```
OSStatus  CreateChasingArrowsControl(WindowRef window,const Rect *boundsRect,
                                     ControlRef *outControl);
```

User Panes — Embedding Control

The user pane has two main uses:

· It can be used as an embedding 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 (callback) 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 32767.
Maximum	Ignored. After user pane creation, reset to a setting between –32768 to 32767.

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.

Programmatic Creation

User Pane controls may be created programmatically using the function `CreateUserPaneControl`:

```
OSStatus  CreateUserPaneControl(WindowRef window,const Rect *boundsRect,
                                UInt32 features,ControlRef *outControl);
```

Scrolling Text Box

The scrolling text box implements a scrolling box of non-editable text. 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`.

Variants and Control Definition IDs

The scrolling text box CDEF resource ID is 27. 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
`KControlScrollTextBoxDelayBefore AutoScrollTag`	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`
`KControlScrollTextBoxDelay BetweenAutoScrollTag`	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`
`KControlScrollTextBoxAutoScroll AmountTag`	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`

Programmatic Creation

Scrolling text box controls may be created programmatically using the function `CreateScrollingTextBoxControl`:

```
OSStatus  CreateScrollingTextBoxControl(WindowRef window,const Rect *boundsRect,
                                        SInt16 contentResID,Boolean autoScroll,
                                        UInt32 delayBeforeAutoScroll,
                                        UInt32 delayBetweenAutoScroll,
                                        UInt16 autoScrollAmount,ControlRef *outControl);
```

Disclosure Button

The disclosure (see Fig 13) button is available only on Mac OS X, and can only be created programmatically using the function `CreateDisclosureButtonControl`:

```
OSStatus  CreateDisclosureButtonControl(WindowRef inWindow,const Rect *inBoundsRect,
                                        SInt32 inValue,Boolean inAutoToggles,
                                        ControlRef *  outControl);
```

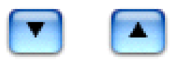

FIG 13 - DISCLOSURE BUTTON

The height of the button is fixed at 20 pixels. The button will be as wide as the specified rectangle and will be centred in that rectangle vertically.

Relevant constants are:

Parameter	***Constants***
`inValue`	`kControlDisclosureButtonClosed` `kControlDisclosureButtonDisclosed`

Edit Unicode Text

The edit Unicode text control is available only on Mac OS X, and can only be created programmatically using the function `CreateEditUnicodeTextControl`:

```
OSStatus  CreateEditUnicodeTextControl(WindowRef window,const Rect *boundsRect,
                                       CFStringRef text,Boolean isPassword,
                                       const ControlFontStyleRec *style,
                                       ControlRef *outControl);
```

The control data tags applicable to the edit text control also apply to the edit Unicode text control, except that the `kControlEditTextTEHandleTag` tag must not be used.

Round Button

The round button is available only on Mac OS X, and can only be created programmatically using the function `CreateRoundButtonControl`:

```
OSStatus  CreateRoundButtonControl(WindowRef nWindow,const Rect *inBoundsRect,
                                   ControlRoundButtonSize inSize,
                                   ControlButtonContentInfo *inContent,
                                   ControlRef *outControl);
```

Relevant constants are:

Parameter	***Constants***
`inSize`	`kControlRoundButtonNormalSize` (20-pixel diameter) `kControlRoundButtonLargeSize` (25-pixel diameter)

Control Data Tag Constants

Control Data Tag Constant	***Meaning and Data Type Returned or Set***
`kControlRoundButtonContentTag`	Sets button content. **Data type set:** `ControlButtonContentInfo`
`kControlRoundButtonSizeTag`	Sets button size. **Data type set:** `ControlRoundButtonSize`

Small Versions of Controls

Human Interface Guidelines permit the use of small versions of certain controls, which should only be used when space is at a premium. Full size and small controls should not be mixed in the same window.

The following lists those controls described in this chapter that are available in small versions, and describes how to create them.

Control	Mac OS X	Mac OS 8/9
Tab	Pass `kControlTabSizeSmall` in `CreateTabsControl` call or use a small variant if creating from a `'CNTL'` resource. Alternatively, if the control is created large, call `SetControlData` with the `kControlSizeTag` tag.	Pass `kControlTabSizeSmall` in `CreateTabsControl` call or use a small variant if creating from a `'CNTL'` resource.
Edit text	Make the control's rectangle 13 pixels high and call `SetControlFontStyle` to set the control's font to the small system font.	Make the control's rectangle 13 pixels high and call `SetControlFontStyle` to set the control's font to the small system font.
Slider	Call `SetControlData` with the `kControlSizeTag` tag.	(Not available.)
Pop-up arrow	Pass `kControlPopupArrowSizeSmall` in `CreatePopupArrowControl`call or use a small variant if creating from a `'CNTL'` resource. Alternatively, if the control is created large, call `SetControlData` with the `kControlSizeTag` to make it small.	Pass `kControlPopupArrowSizeSmall` in `CreatePopupArrowControl`call or use a small variant if creating from a `'CNTL'` resource.
Round button	Call `SetControlData` with the `kControlSizeTag` tag.	(Not available.)

Idle Processing

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

Control	Reason For Idle Processing
Progress bar (indeterminate variant).	Animate the indicator.
Clocks (when the `kControlClockIsLive` clock value flag constant is used).	Update the clock.
Chasing arrows.	Animate the arrows.
Edit text controls.	Cause TextEdit to be called to blink the insertion point caret.
Scrolling text box (auto-scrolling variant).	Scroll the text automatically.

When these controls are displayed in a document window, your application may need to call `IdleControls` at the appropriate interval in your event loop, otherwise the animations, etc., will not occur. `IdleControls` sends a particular message to the CDEF, which responds appropriately.

The call to `IdleControls` is not necessary on Mac OS X because Mac OS X controls idle themselves automatically using their own internal timers. However, as of the time of writing, the call is necessary on Mac OS 8/9 because controls on Mac OS 8/9 do not yet have internal timers.

When these controls are displayed in a dialog on Mac OS 8/9, `ModalDialog` calls `IdleControls` for you.

You can determine whether a control has an internal timer by calling `GetControlFeatures` and testing for the `kControlIdlesWithTimer` bit.

Defining Your Own Key Filter Function

You can attach a **key filter function** to edit text controls 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.

You would declare your key filter function as follows:

```
ControlKeyFilterResult  myKeyFilterFunction(ControlRef controlRef,SInt16* keyCode,
                                            SInt16 *charCode,EventModifiers *modifiers);
```

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

```
typedef STACK_UPP_TYPE(ControlKeyFilterProcPtr) ControlKeyFilterUPP;
```

As stated at Edit Text, 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 control and assumes a key filter function named `numericFilter`.

```
#define kLeftArrow  0x1C
#define kRightArrow 0x1D
#define kUpArrow    0x1E
#define kDownArrow  0x1F
#define kBackspace  0x08
...

ControlKeyFilterUPP numericFilterUPP;
ControlRef          controlHdl;
...

// .................................................................... get universal procedure pointer

numericFilterUPP = NewControlKeyFilterProc(numericFilter);

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

GetDialogItemAsControl(dialogPtr,itemNumber,&controlHdl);
SetControlData(controlHdl,kControlNoPart,kControlEditTextKeyFilterTag,
               sizeof(numericFilterUPP),&numericFilterUPP);

...

// ********************************************************************************* numericFilter
//
// This function will be called each time the edit text control 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.

ControlKeyFilterResult numericFilter(ControlRef control,SInt16 *keyCode,
                                     SInt16 *charCode,SInt16 *modifiers)
{
  if(((char) *charCode >= '0') && ((char) *charCode <= '9'))
    return kControlKeyFilterPassKey;

  switch(*charCode)
  {
    case '-':
    case kLeftArrow:
    case kRightArrow:
    case kUpArrow:
    case kDownArrow:
    case kBackspace:
      return kControlKeyFilterPassKey;
      break;
    default:
      SysBeep(10);
      return kControlKeyFilterBlockKey;
      break;
  }
}
```

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.

You would declare your edit text validation function as follows:

```
void  myEditTextValidationFunction(ControlRef controlHdl);
```

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.

```
ControlEditTextValidationUPP editTextValidatorUPP;
ControlRef                   controlHdl;
...

// ………………………………………………………………………………………………… get universal procedure pointer

editTextValidatorUPP  = NewControlEditTextValidationProc(editTextValidator);

// …………………………………………………………………………… attach edit text validation function to the control

GetDialogItemAsControl(dialogPtr,iEditText1,&controlHdl);
SetControlData(controlHdl,kControlNoPart,kControlEditTextValidationProcTag,
               sizeof(editTextValidatorUPP),&editTextValidatorUPP);
...

void  editTextValidator(ControlRef controlHdl)
{
  Str31 theText;
  Size  actualSize;
  UInt8 a;

  // …………………………………………………………………………… Get the text to be examined from the control

  GetControlData(controlHdl,kControlNoPart,kControlEditTextTextTag,sizeof(theText) -1,
                 (Ptr) &theText[1],&actualSize);

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

  if(actualSize <= 31)
    theText[0] = actualSize;
  else
    theText[0] = 31;

  // ………………………………………………………………………………………………………… Replace any colons with dashes

  for(a=1;a<=theText[0];a++)
  {
    if(theText[a] == ':')
      theText[a] = '-';
  }

  // …………………………………………………………………………………………………………… 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,theText[0],
                 (Ptr) &theText[1]);

  Draw1Control(controlHdl);
}
```

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 (callback) functions which perform actions such as drawing, hit testing, tracking, etc. Such application-defined (callback) functions are called **user pane functions**. User pane functions provide you with the ability to create a custom control without writing your own control definition function.

User Pane Functions

User pane functions are categorised as follows:

Function	*Description*
Draw	Draws the content of a user pane control.
Hit test	Returns the part code when a mouse-down event occurs.
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 Data Tag Constants and Universal Procedure Pointers

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 set passed in the `inTagName` parameter and a universal procedure pointer to the user pane function passed in the `inData` parameter.

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 drawing function. Indicates that the Control Manager needs to draw a control. **Data type returned or set:** `ControlUserPaneDrawingUPP`
`kControlUserPaneHitTestProcTag`	Gets or sets a 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 tracking function. The tracking function 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 an idle function. The idle function 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 key down function. The key down function 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 an activate function. The activate function 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 keyboard focus function. The keyboard focus function 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 background function. The background function 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`

Creating and Disposing of Universal Procedure Pointers

The following functions create and dispose of universal procedure pointers for user pane functions:

```
NewControlUserPaneDrawUPP       DisposeControlUserPaneDrawUPP
NewControlUserPaneHitTestUPP    DisposeControlUserPaneHitTestUPP
NewControlUserPaneTrackingUPP   DisposeControlUserPaneTrackingUPP
NewControlUserPaneIdleUPP       DisposeControlUserPaneIdleUPP
NewControlUserPaneKeyDownUPP    DisposeControlUserPaneKeyDownUPP
NewControlUserPaneActivateUPP   DisposeControlUserPaneActivateUPP
NewControlUserPaneFocusUPP      DisposeControlUserPaneFocusUPP
NewControlUserPaneBackgroundUPP DisposeControlUserPaneBackgroundUPP
```

Main Constants, Data Types and Functions

Constants

Control Definition IDs

```
KControlBevelButtonSmallBevelProc              = 32
KControlBevelButtonNormalBevelProc             = 33
KControlBevelButtonLargeBevelProc              = 34
kControlSliderProc                             = 48
kControlSliderLiveFeedback                     = (1 << 0)
kControlSliderHasTickMarks                     = (1 << 1)
kControlSliderReverseDirection                 = (1 << 2)
kControlSliderNonDirectional                   = (1 << 3)
kControlTriangleProc                           = 64
kControlTriangleLeftFacingProc                 = 65
kControlTriangleAutoToggleProc                 = 66
kControlTriangleLeftFacingAutoToggleProc       = 67
kControlProgressBarProc                        = 80
kControlLittleArrowsProc                       = 96
kControlChasingArrowsProc                      = 112
kControlTabLargeNorthProc                      = 128
kControlTabSmallNorthProc                      = 129
kControlTabLargeSouthProc                      = 130
kControlTabSmallSouthProc                      = 131
kControlTabLargeEastProc                       = 132
kControlTabSmallEastProc                       = 133
kControlTabLargeWestProc                       = 134
kControlTabSmallWestProc                       = 135
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	= 1 << 3

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	= 0x0100
kControlBehaviorSticky	= 0x0200
kControlBehaviorSingleValueMenu	= 0

```
kControlBehaviorMultiValueMenu                = 0x4000
kControlBehaviorOffsetContents                = 0x8000
kControlBehaviorCommandMenu                   = 0x2000
```

Bevel Button Content Type

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

Bevel Button Thickness

```
kControlBevelButtonSmallBevel                 = 0
kControlBevelButtonNormalBevel                = 1
kControlBevelButtonLargeBevel                 = 2
```

Bevel Button Menu Placement

```
kControlBevelButtonMenuOnBottom               = 0
kControlBevelButtonMenuOnRight                = (1 << 2)
```

Tab Size

```
kControlTabSizeLarge                          = 0
kControlTabSizeSmall                          = 1
```

Tab Direction

```
kControlTabDirectionNorth                     = 0
kControlTabDirectionSouth                     = 1
kControlTabDirectionEast                      = 2
kControlTabDirectionWest                      = 3
```

Clock Value Flag Constants

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

Control Data Tags

```
kControlFontStyleTag                          = FOUR_CHAR_CODE('font')
kControlKeyFilterTag                          = FOUR_CHAR_CODE('fltr')
kControlBevelButtonContentTag                 = FOUR_CHAR_CODE('cont')
kControlBevelButtonTransformTag               = FOUR_CHAR_CODE('tran')
kControlBevelButtonTextAlignTag               = FOUR_CHAR_CODE('tali')
kControlBevelButtonTextOffsetTag              = FOUR_CHAR_CODE('toff')
kControlBevelButtonGraphicAlignTag            = FOUR_CHAR_CODE('gali')
kControlBevelButtonGraphicOffsetTag           = FOUR_CHAR_CODE('goff')
kControlBevelButtonTextPlaceTag               = FOUR_CHAR_CODE('tplc')
kControlBevelButtonMenuValueTag               = FOUR_CHAR_CODE('mval')
kControlBevelButtonMenuRefTag                 = FOUR_CHAR_CODE('mhnd')
kControlBevelButtonOwnedMenuRefTag            = FOUR_CHAR_CODE('omrf')
kControlBevelButtonKindTag                    = FOUR_CHAR_CODE('bebk')
kControlBevelButtonCenterPopupGlyphTag        = FOUR_CHAR_CODE('pglc')
kControlBevelButtonLastMenuTag                = FOUR_CHAR_CODE('lmnu')
kControlBevelButtonMenuDelayTag               = FOUR_CHAR_CODE('mdly')
kControlBevelButtonScaleIconTag               = FOUR_CHAR_CODE('scal')
kControlTriangleLastValueTag                  = FOUR_CHAR_CODE('last')
kControlProgressBarIndeterminateTag           = FOUR_CHAR_CODE('inde')
kControlTabContentRectTag                     = FOUR_CHAR_CODE('rect')
kControlTabEnabledFlagTag                     = FOUR_CHAR_CODE('enab')
kControlTabFontStyleTag                       = kControlFontStyleTag
kControlTabInfoTag                            = FOUR_CHAR_CODE('tabi')
kControlGroupBoxMenuHandleTag                 = FOUR_CHAR_CODE('mhan')
kControlGroupBoxFontStyleTag                  = kControlFontStyleTag
kControlGroupBoxTitleRectTag                  = FOUR_CHAR_CODE('trec')
kControlImageWellContentTag                   = FOUR_CHAR_CODE('cont')
kControlImageWellTransformTag                 = FOUR_CHAR_CODE('tran')
kControlClockLongDateTag                      = FOUR_CHAR_CODE('date')
```

```
kControlClockFontStyleTag                       = kControlFontStyleTag
kControlClockAnimatingTag                       = FOUR_CHAR_CODE('anim')
kControlUserItemDrawProcTag                     = FOUR_CHAR_CODE('uidp')
kControlUserPaneDrawProcTag                     = FOUR_CHAR_CODE('draw')
kControlUserPaneHitTestProcTag                  = FOUR_CHAR_CODE('hitt')
kControlUserPaneTrackingProcTag                 = FOUR_CHAR_CODE('trak')
kControlUserPaneIdleProcTag                     = FOUR_CHAR_CODE('idle')
kControlUserPaneKeyDownProcTag                  = FOUR_CHAR_CODE('keyd')
kControlUserPaneActivateProcTag                 = FOUR_CHAR_CODE('acti')
kControlUserPaneFocusProcTag                    = FOUR_CHAR_CODE('foci')
kControlUserPaneBackgroundProcTag               = FOUR_CHAR_CODE('back')
kControlEditTextStyleTag                        = kControlFontStyleTag
kControlEditTextTextTag                         = FOUR_CHAR_CODE('text')
kControlEditTextTEHandleTag                     = FOUR_CHAR_CODE('than')
kControlEditTextKeyFilterTag                    = kControlKeyFilterTag,
kControlEditTextSelectionTag                    = FOUR_CHAR_CODE('sele')
kControlEditTextPasswordTag                     = FOUR_CHAR_CODE('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')
kControlRoundButtonContentTag                   = FOUR_CHAR_CODE('cont'
kControlRoundButtonSizeTag                      = FOUR_CHAR_CODE('size'
```

Slider Orientation

```
kControlSliderPointsDownOrRight                 = 0
kControlSliderPointsUpOrLeft                    = 1
kControlSliderDoesNotPoint                      = 2
```

Clock Control Type

```
kControlClockTypeHourMinute                     = 0
kControlClockTypeHourMinuteSecond               = 1
kControlClockTypeMonthDayYear                   = 2
kControlClockTypeMonthYear                      = 3
```

Disclosure Triangle Orientation

```
kControlDisclosureTrianglePointDefault          = 0
kControlDisclosureTrianglePointRight            = 1
kControlDisclosureTrianglePointLeft             = 2
```

Pop-up Arrow Orientation

```
kControlPopupArrowOrientationEast               = 0
kControlPopupArrowOrientationWest               = 1
kControlPopupArrowOrientationNorth              = 2
kControlPopupArrowOrientationSouth              = 3
```

Key Filter Result Codes

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

Control Feature Bits

```
kControlSupportsGhosting                        = 1 << 0
kControlSupportsEmbedding                       = 1 << 1
kControlSupportsFocus                           = 1 << 2
```

```
kControlWantsIdle                           = 1 << 3
kControlWantsActivate                       = 1 << 4
kControlHandlesTracking                     = 1 << 5
kControlSupportsDataAccess                  = 1 << 6
kControlHasSpecialBackground                = 1 << 7
kControlGetsFocusOnClick                    = 1 << 8
kControlSupportsCalcBestRect                = 1 << 9
kControlSupportsLiveFeedback                = 1 << 10
kControlHasRadioBehavior                    = 1 << 11
```

Focusing Part Codes

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

Disclosure Button State

```
kControlDisclosureButtonClosed              = 0
kControlDisclosureButtonDisclosed           = 1
```

Round Button Size

```
kControlRoundButtonNormalSize               = 0
kControlRoundButtonLargeSize                = 2
```

Control Feature Bit – Internal Timer

```
kControlIdlesWithTimer                      = 1 << 23
```

Control Kind (Mac OS X Only)

```
kControlKindBevelButton                     = FOUR_CHAR_CODE('bevl')
kControlKindImageWell                       = FOUR_CHAR_CODE('well')
kControlKindTabs                            = FOUR_CHAR_CODE('tabs')
kControlKindEditText                        = FOUR_CHAR_CODE('etxt')
kControlKindSlider                          = FOUR_CHAR_CODE('sldr')
kControlKindCheckGroupBox                   = FOUR_CHAR_CODE('cgrp')
kControlKindPopupGroupBox                   = FOUR_CHAR_CODE('pgrp')
kControlKindClock                           = FOUR_CHAR_CODE('clck')
kControlKindProgressBar                     = FOUR_CHAR_CODE('prgb')
kControlKindRelevanceBar                    = FOUR_CHAR_CODE('relb')
kControlKindLittleArrows                    = FOUR_CHAR_CODE('larr')
kControlKindDisclosureTriangle              = FOUR_CHAR_CODE('dist')
kControlKindPicture                         = FOUR_CHAR_CODE('pict')
kControlKindIcon                            = FOUR_CHAR_CODE('icon')
kControlKindWindowHeader                    = FOUR_CHAR_CODE('whed')
kControlKindPlacard                         = FOUR_CHAR_CODE('plac')
kControlKindStaticText                      = FOUR_CHAR_CODE('stxt')
kControlKindSeparator                       = FOUR_CHAR_CODE('sepa')
kControlKindPopupArrow                      = FOUR_CHAR_CODE('parr')
kControlKindRadioGroup                      = FOUR_CHAR_CODE('rgrp')
kControlKindChasingArrows                   = FOUR_CHAR_CODE('carr')
kControlKindScrollingTextBox                = FOUR_CHAR_CODE('stbx')
kControlKindDisclosureButton                = FOUR_CHAR_CODE('disb')
kControlKindRoundButton                     = FOUR_CHAR_CODE('rndb')
```

Data Types

```
typedef SInt16 ControlFocusPart;
typedef SInt16 ControlKeyFilterResult;
typedef SInt16 ControlButtonGraphicAlignment;
typedef SInt16 ControlButtonTextAlignment;
typedef SInt16 ControlButtonTextPlacement;
typedef SInt16 ControlContentType;
typedef SInt16 ControlVariant;
typedef UInt16 ControlSliderOrientation;
typedef UInt16 ControlClockType;
typedef UInt16 ControlDisclosureTriangleOrientation;
typedef SInt16 ControlRoundButtonSize;
```

Button Content Info Structure

```
struct ControlButtonContentInfo
{
  ControlContentType  contentType;
  union {
    SInt16      resID;
    CIconHandle cIconHandle;
    Handle      iconSuite;
    IconRef     iconRef;
    PicHandle   picture;
    Handle      ICONHandle;
  } u;
};
typedef struct ControlButtonContentInfo ControlButtonContentInfo;
typedef ControlButtonContentInfo *ControlButtonContentInfoPtr;
typedef ControlButtonContentInfo ControlImageContentInfo;
typedef ControlButtonContentInfo *ControlImageContentInfoPtr;
```

Tab Information Structures

```
struct ControlTabInfoRec
{
  SInt16 version;
  SInt16 iconSuiteID;
  Str255 name;
};
typedef struct ControlTabInfoRec ControlTabInfoRec;

struct ControlTabInfoRecV1
{
  SInt16      version;
  SInt16      iconSuiteID;
  CFStringRef name;
};
typedef struct ControlTabInfoRecV1 ControlTabInfoRecV1;
```

Tab Entry Structures

```
struct ControlTabEntry
{
  ControlButtonContentInfo *icon;
  CFStringRef               name;
  Boolean                   enabled;
};
typedef struct ControlTabEntry ControlTabEntry;
```

Edit Text Selection Structure

```
struct ControlEditTextSelectionRec
{
  SInt16 selStart;
  SInt16 selEnd;
};
typedef struct ControlEditTextSelectionRec ControlEditTextSelectionRec;
typedef ControlEditTextSelectionRec * ControlEditTextSelectionPtr;
```

Functions

Give Idle Time To Controls

```
void     IdleControls(WindowPtr inWindow);
```

Send Keyboard Event to Control With keyboard Focus

```
Sint16    HandleControlKey(ControlRef inControl,SInt16 inKeyCode,SInt16 inCharCode,
          SInt16 inModifiers);
```

Set the Background For a Control

```
OSErr     SetUpControlBackground (ControlRef inControl,SInt16 inDepth,Boolean inIsColorDevice);
```

KeyBoard Focus

```
OSErr     GetKeyboardFocus(WindowPtr inWindow,ControlRef *outControl);
OSErr     SetKeyboardFocus(WindowPtr inWindow,ControlRef inControl,ControlFocusPart inPart);
OSErr     AdvanceKeyboardFocus(WindowPtr inWindow);
OSErr     ReverseKeyboardFocus(WindowPtr inWindow);
OSErr     ClearKeyboardFocus(WindowPtr inWindow);
```

Control Features

```
OSErr     GetControlFeatures(ControlRef inControl,UInt32 *outFeatures);
```

Validating Controls

```
Boolean   IsValidControlhandle(ControlRef theControl);
```

Creating Controls Programmatically

```
OSStatus  CreateBevelButtonControl(WindowRef window,const Rect *boundsRect,
          CFStringRef title,ControlBevelThickness thickness,
          ControlBevelButtonBehavior behavior,ControlButtonContentInfoPtr info, SInt16 menuID,
          ControlBevelButtonMenuBehavior menuBehavior,
          ControlBevelButtonMenuPlacement menuPlacement,ControlRef *outControl);
OSStatus  CreateImageWellControl(WindowRef window,const Rect *boundsRect,
          const ControlButtonContentInfo *info,ControlRef *outControl);
OSStatus  CreateTabsControl(WindowRef window,const Rect *boundsRect,ControlTabSize size,
          ControlTabDirection direction,UInt16 numTabs,const ControlTabEntry *tabArray,
          ControlRef *outControl);
OSStatus  CreateEditTextControl(WindowRef window,const Rect *boundsRect, CFStringRef text,
          Boolean isPassword,Boolean useInlineInput,const ControlFontStyleRec *style,
          ControlRef *outControl);
OSStatus  CreateSliderControl(WindowRef window,const Rect *boundsRect,SInt32 value,
          SInt32 minimum,SInt32 maximum,ControlSliderOrientation  orientation,
          UInt16 numTickMarks,Boolean liveTracking,ControlActionUPP liveTrackingProc,
          ControlRef *outControl);
OSStatus  CreateGroupBoxControl(WindowRef window,const Rect *boundsRect, CFStringRef title,
          Boolean primary,ControlRef *outControl);
OSStatus  CreateCheckGroupBoxControl(WindowRef window,const Rect *boundsRect,
          CFStringRef title,Boolean primary,Boolean autoToggle,ControlRef *outControl);
OSStatus  CreatePopupGroupBoxControl(WindowRef window,const Rect *boundsRect,
          CFStringRef title,Boolean primary,SInt16 menuID,Boolean variableWidth,
          SInt16 titleWidth,SInt16 titleJustification,Style titleStyle,ControlRef *outControl);
OSStatus  CreateClockControl(WindowRef window,const Rect *boundsRect,
          ControlClockType clockType,
          ControlClockFlags clockFlags,ControlRef *outControl);
OSStatus  CreateProgressBarControl(WindowRef window,const Rect *boundsRect,SInt32 value,
          SInt32 minimum,SInt32 maximum,Boolean indeterminate,ControlRef *outControl);
OSStatus  CreateLittleArrowsControl(WindowRef window,const Rect *boundsRect,SInt32 value,
          SInt32 minimum,SInt32 maximum,SInt32 increment,ControlRef *outControl);
OSStatus  CreateDisclosureTriangleControl(WindowRef window,const Rect *boundsRect,
          ControlDisclosureTriangleOrientation orientation, CFStringRef title,
          Boolean drawTitle,Boolean autoToggles,ControlRef *outControl);
OSStatus  CreatePictureControl(WindowRef window,const Rect *boundsRect,
          const ControlButtonContentInfo *content,Boolean dontTrack,ControlRef *outControl);
OSStatus  CreateIconControl(WindowRef window,const Rect *boundsRect,
          const ControlButtonContentInfo *icon,Boolean dontTrack,ControlRef *outControl);
OSStatus  CreateWindowHeaderControl(WindowRef window,const Rect *boundsRect,
          Boolean isListHeader,ControlRef *outControl);
OSStatus  CreatePlacardControl(WindowRef window,const Rect *boundsRect,ControlRef *outControl);
OSStatus  CreateStaticTextControl(WindowRef window,const Rect *boundsRect,
          CFStringRef text,const ControlFontStyleRec *style,ControlRef *outControl);
OSStatus  CreateSeparatorControl(WindowRef window,const Rect *boundsRect,
          ControlRef *outControl);
OSStatus  CreatePopupArrowControl(WindowRef window,const Rect *boundsRect,
          ControlPopupArrowOrientation orientation,ControlPopupArrowSize size,
          ControlRef *outControl);
OSStatus  CreateRadioGroupControl(WindowRef window,const Rect *boundsRect,
          ControlRef *outControl);
OSStatus  CreateChasingArrowsControl(WindowRef window,const Rect *boundsRect,
          ControlRef *outControl);
OSStatus  CreateUserPaneControl(WindowRef window,const Rect *boundsRect,UInt32 features,
          ControlRef *outControl);
```

```
OSStatus  CreateScrollingTextBoxControl(WindowRef window,const Rect *boundsRect,
          SInt16 contentResID, Boolean autoScroll,UInt32 delayBeforeAutoScroll,
          UInt32 delayBetweenAutoScroll, UInt16 autoScrollAmount,ControlRef *outControl);
```

Creating Controls Programmatically (Controls Available on Mac OS X Only)

```
OSStatus  CreateRelevanceBarControl(WindowRef window,const Rect *boundsRect,SInt32 value,
          SInt32 minimum,SInt32 maximum,ControlRef *outControl);
OSStatus  CreateDisclosureButtonControl(WindowRef inWindow,const Rect *inBoundsRect,
          SInt32 inValue,Boolean inAutoToggles,ControlRef *  outControl);
OSStatus  CreateEditUnicodeTextControl(WindowRef window,const Rect *boundsRect,
          CFStringRef text,Boolean isPassword,const ControlFontStyleRec *style,
          ControlRef *outControl);
OSStatus  CreateRoundButtonControl(WindowRef nWindow,const Rect *inBoundsRect,
          ControlRoundButtonSize inSize,ControlButtonContentInfo *inContent,
          ControlRef *outControl);
```

Helper Functions

```
OSErr     GetBevelButtonMenuValue ControlRef inButton,SInt16 * outValue);
OSErr     SetBevelButtonMenuValue ControlRef inButton,SInt16 inValue);
OSErr     GetBevelButtonMenuHandle ControlRef inButton,MenuHandle * outHandle);
OSErr     GetBevelButtonContentInfo ControlRef inButton,
          ControlButtonContentInfoPtr outContent);
OSErr     SetBevelButtonContentInfo ControlRef inButton,ControlButtonContentInfoPtr inContent);
OSErr     SetBevelButtonTransform ControlRef inButton,IconTransformType transform);
OSErr     SetBevelButtonGraphicAlignment ControlRef inButton,
          ControlButtonGraphicAlignment inAlign,SInt16 inHOffset,SInt16 inVOffset);
OSErr     SetBevelButtonTextAlignment ControlRef inButton,ControlButtonTextAlignment inAlign,
          SInt16 inHOffset);
OSErr     SetBevelButtonTextPlacement ControlRef inButton,ControlButtonTextPlacement inWhere);
OSErr     GetImageWellContentInfo(ControlRef inButton,ControlButtonContentInfoPtr outContent);
OSErr     SetImageWellContentInfo(ControlRef inButton,ControlButtonContentInfoPtr inContent);
OSErr     SetImageWellTransform(ControlRef inButton,IconTransformType inTransform);
OSErr     GetTabContentRect(ControlRef inTabControl,Rect * outContentRect);
OSErr     SetTabEnabled(ControlRef inTabControl,SInt16 inTabToHilite,Boolean inEnabled);
OSErr     SetDisclosureTriangleLastValue(ControlRef inTabControl,SInt16 inValue);
```

Application-Defined (Callback) Functions

```
ControlKeyFilterResult  myKeyFilterFunction(ControlRef controlRef,SInt16* keyCode,
                         SInt16 *charCode,EventModifiers *modifiers);
void                    myEditTextValidationFunction(ControlRef controlHdl);
```

Creating and Disposing of Universal Procedure Pointers — User Pane Functions

```
ControlUserPaneDrawUPP       NewControlUserPaneDrawUPP(ControlUserPaneDrawProcPtr userRoutine);
ControlUserPaneHitTestUPP    NewControlUserPaneHitTestUPP(ControlUserPaneHitTestProcPtr
                             userRoutine);
ControlUserPaneTrackingUPP   NewControlUserPaneTrackingUPP(ControlUserPaneTrackingProcPtr
                             userRoutine);
ControlUserPaneIdleUPP       NewControlUserPaneIdleUPP(ControlUserPaneIdleProcPtr
                             userRoutine);
ControlUserPaneKeyDownUPP    NewControlUserPaneKeyDownUPP(ControlUserPaneKeyDownProcPtr
                             userRoutine);
ControlUserPaneActivateUPP   NewControlUserPaneActivateUPP(ControlUserPaneActivateProcPtr
                             userRoutine);
ControlUserPaneFocusUPP      NewControlUserPaneFocusUPP(ControlUserPaneFocusProcPtr
                             userRoutine);
ControlUserPaneBackgroundUPP NewControlUserPaneBackgroundUPP(ControlUserPaneBackgroundProcPtr
                             userRoutine);
void      DisposeControlUserPaneDrawUPP(ControlUserPaneDrawUPP userUPP);
void      DisposeControlUserPaneHitTestUPP(ControlUserPaneHitTestUPP userUPP);
void      DisposeControlUserPaneTrackingUPP(ControlUserPaneTrackingUPP userUPP);
void      DisposeControlUserPaneIdleUPP(ControlUserPaneIdleUPP userUPP);
void      DisposeControlUserPaneKeyDownUPP(ControlUserPaneKeyDownUPP userUPP);
void      DisposeControlUserPaneActivateUPP(ControlUserPaneActivateUPP userUPP);
void      DisposeControlUserPaneFocusUPP(ControlUserPaneFocusUPP userUPP);
void      DisposeControlUserPaneBackgroundUPP(ControlUserPaneBackgroundUPP userUPP);
```

Demonstration Program Controls3 Listing

```
// *******************************************************************************************
// Controls3.h                                                           CLASSIC EVENT MODEL
// *******************************************************************************************
//
// 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:
//
// •  A 'plst' resource.
//
// •  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' resources (purgeable).
//
// •  'CNTL' resources (purgeable).
//
// •  A 'tab#' resource (purgeable.
//
// •  An icon family resource (purgeable).
//
// •  'PICT' resources (purgeable).
//
// •  'cicn' resources (purgeable).
//
// •  'STR#' resources (purgeable).
//
// •  'TEXT' and 'styl' resources (purgeable).
//
// •  'hrct' and an 'hwin' resources (preload, purgeable), which provide help balloons
//    describing  the various controls.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// *******************************************************************************************

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

#include <Carbon.h>
#include <string.h>

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

#define rMenubar                128
#define mAppleApplication       128
#define  iAbout                 1
#define mFile                   129
#define  iQuit                  12
#define mDemonstration          131
#define  iBevelAndImage          1
#define  iTabEditClock          2
#define  iGroupArrowsProgress   3
#define  iSliders               4
#define  iTextBoxes             5
#define  iSmallControls         6
#define rBevelImageWindow       128
#define  cBevelButton1          128
#define  cBevelButton2          129
#define  cBevelButton3          130
```

```
#define  cBevelButton4          131
#define  cBevelButton5          132
#define  cBevelButton6          133
#define  cBevelButton7          134
#define  cBevelButton8          135
#define  cBevelButton9          136
#define  cBevelButton10         137
#define  cBevelButton11         138
#define  cBevelButton12         139
#define  cBevelButton13         140
#define  cBevelButton14         141
#define  cBevelButton15         142
#define  cBevelButton16         143
#define  cBevelButton17         144
#define  cBevelButton18         145
#define  cBevelButton19         146
#define  cBevelButton20         147
#define  cBevelButton21         148
#define  cImageWell1            149
#define  cImageWell2            150
#define  cPicture1              151
#define  cPicture2              152
#define  cColourIcon1           153
#define  cColourIcon2           154
#define  cIconSuite1            155
#define  cIconSuite2            156
#define  cWindowHeader          157
#define  rPartCodeStrings       128
#define  rGraphicAlignStrings   129
#define  rTextAlignStrings      130
#define  rTextPlacementStrings  131
#define rTabEditClockDialog     128
#define  iTabs                  2
#define  tabEditText            1
#define  tabClocks              2
#define  iEditTextUserPane      3
#define  iEditText1             5
#define  iEditText2             7
#define  iEditText3             9
#define  iExtractEditText       10
#define  iClocksUserPane        12
#define  iImageWellEditText     11
#define  iClocks1               14
#define  iClocks2               16
#define  iClocks3               18
#define  iExtractClocks         19
#define  iImageWellClocks       20
#define  kLeftArrow             0x1C
#define  kRightArrow            0x1D
#define  kUpArrow               0x1E
#define  kDownArrow             0x1F
#define  kBackspace             0x08
#define  kDelete                0x7F
#define rGroupArrowsProgDialog  129
#define  iCheckboxGroup         2
#define  iRadioGroupColour      3
#define  iStaticTextColourDepth 7
#define  iPopupGroup            8
#define  iUserPaneNamesInitials 9
#define  iRadioGroupNames       10
#define  iCheckboxShowInitials  13
#define  iUserPaneScoreAverage  15
#define  iRadioGroupScores      16
#define  iCheckboxShowAverages  19
#define  iStaticTextCache       26
#define  iLittleArrows          27
#define  iPushButtonExtract     28
#define  iImageWell             29
#define  iDisclosureTriangle    31
```

```
#define  iStaticTextDisclosure  32
#define  iProgressBar           34
#define rSlidersDialog          131
#define  iSlider1               2
#define  iSlider2               3
#define  iSlider3               4
#define  iSlider4               5
#define  iSlider1StaticText     9
#define  iSlider2StaticText     11
#define  iSlider3StaticText     13
#define  iSlider4StaticText     15
#define  iSlider5               17
#define  iUserPane1             18
#define  iSlider6               19
#define cSmallPopup             203
#define rSmallControlsString    136
#define rAboutDialog            132
#define MAX_UINT32              0xFFFFFFFF
#define MIN(a,b)                ((a) < (b) ? (a) : (b))

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

typedef struct
{
  ControlRef bevelButton1Ref;
  ControlRef bevelButton2Ref;
  ControlRef bevelButton3Ref;
  ControlRef bevelButton4Ref;
  ControlRef bevelButton5Ref;
  ControlRef bevelButton6Ref;
  ControlRef bevelButton7Ref;
  ControlRef bevelButton8Ref;
  ControlRef bevelButton9Ref;
  ControlRef bevelButton10Ref;
  ControlRef bevelButton11Ref;
  ControlRef bevelButton12Ref;
  ControlRef bevelButton13Ref;
  ControlRef bevelButton14Ref;
  ControlRef bevelButton15Ref;
  ControlRef bevelButton16Ref;
  ControlRef bevelButton17Ref;
  ControlRef bevelButton18Ref;
  ControlRef bevelButton19Ref;
  ControlRef bevelButton20Ref;
  ControlRef bevelButton21Ref;
  ControlRef imageWell1Ref;
  ControlRef imageWell2Ref;
  ControlRef picture1Ref;
  ControlRef picture2Ref;
  ControlRef colourIcon1Ref;
  ControlRef colourIcon2Ref;
  ControlRef iconSuite1Ref;
  ControlRef iconSuite2Ref;
  ControlRef windowHeaderRef;
} BevelDocStruc;

typedef BevelDocStruc **BevelDocStrucHandle;

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

void    main                        (void);
void    doPreliminaries             (void);
OSErr   quitAppEventHandler         (AppleEvent *,AppleEvent *,SInt32);
void    doGetControls               (WindowRef);
void    doEvents                    (EventRecord *);
void    doMouseDown                 (EventRecord *);
void    doMenuChoice                (SInt32);
void    doUpdate                    (EventRecord *);
void    doActivate                  (EventRecord *);
void    doActivateWindow             (WindowRef,Boolean);
void    doConcatPStrings             (Str255,Str255);
void    doCopyPString                (Str255,Str255);

void    doBevelImagePictIcon         (void);
void    doBevelImagePictIconContent  (EventRecord *,WindowRef);
void    doDrawPartCode               (WindowRef,ControlRef,SInt16,SInt16);
void    doGraphicAlignment           (WindowRef,ControlRef,ControlRef);
void    doTextAlignment              (WindowRef,ControlRef,ControlRef);
void    doTextOffset                 (WindowRef,ControlRef,ControlRef);
void    doTextPlacement              (WindowRef,ControlRef,ControlRef);
void    doDrawMessage                (WindowRef,Boolean);
void    doDrawLegends                (WindowRef,Boolean);
void    helpTagsBevelImagePictIcon   (WindowRef);

void    doTabEditClock               (void);
void    doExtractEditText            (DialogRef);
void    doExtractDateTime            (DialogRef);
void    helpTagsTabEditClock         (DialogRef);

void    doGroupArrowsProgress        (void);
void    doCheckBoxGroupBox           (DialogRef);
void    doPopupGroupBox              (DialogRef);
void    doChasingAndProgress         (DialogRef);
void    doExtractCurrentStatus       (DialogRef);
void    helpTagsGroupArrowsProgress  (DialogRef);

void    doSliderUserPane             (void);
void    doDrawSliderValues           (DialogRef,ControlRef);
void    helpTagsSliders              (DialogRef);

void    doTextBox                    (void);

void    doSmallControls              (void);
void    doMouseDownSmallControls     (WindowRef,EventRecord *);

Boolean eventFilter                  (DialogRef,EventRecord *,SInt16 *);
void    editTextValidator            (ControlRef);
ControlKeyFilterResult  numericFilter (ControlRef,SInt16 *,SInt16 *,EventModifiers *);
void    arrowsActionFunction         (ControlRef,SInt16);
void    sliderActionFunction1        (ControlRef,SInt16);
void    sliderActionFunction2        (ControlRef,SInt16);
void    userPaneDrawFunction         (ControlRef,SInt16);
void    userPaneActivateFunction     (ControlRef,Boolean);

// *******************************************************************************************
// Controls3.c
// *******************************************************************************************

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

#include "Controls3.h"

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

Boolean gRunningOnX                = false;
Boolean gInBackground              = false;
Boolean gDone;
Str255  gCurrentString;
Boolean gBevelAndImageActive       = false;
Boolean gGroupArrowsProgressActive = false;
Boolean gSlidersActive             = false;

// ************************************************************************************** main

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  EventRecord   eventStructure;

  // ........................................................................ do prelimiaries

  doPreliminaries();

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

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

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }

    menuRef = GetMenuRef(mDemonstration);
    if(menuRef != NULL)
      EnableMenuItem(menuRef,iSmallControls);

    gRunningOnX = true;
  }

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

  gDone = false;

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&eventStructure,MAX_UINT32,NULL))
      doEvents(&eventStructure);
  }
}

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

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(512);
  InitCursor();
  FlushEvents(everyEvent,0);

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

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

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;
  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,NULL,0,
                              &actualSize);

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

  return osError;
}

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

void  doEvents(EventRecord *eventStrucPtr)
{
  SInt32 menuChoice;
  SInt16 menuID, menuItem;

  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case keyDown:
      if((eventStrucPtr->modifiers & cmdKey) != 0)
      {
        menuChoice = MenuEvent(eventStrucPtr);
        menuID = HiWord(menuChoice);
        menuItem = LoWord(menuChoice);
        if(menuID == mFile && menuItem  == iQuit)
          gDone = true;
      }
      break;

    case mouseDown:
      doMouseDown(eventStrucPtr);
      break;

    case updateEvt:
      doUpdate(eventStrucPtr);
      break;

    case activateEvt:
      doActivate(eventStrucPtr);
      break;

    case osEvt:
      switch((eventStrucPtr->message >> 24) & 0x000000FF)
      {
        case suspendResumeMessage:
          if((eventStrucPtr->message & resumeFlag) == 1)
          {
            SetThemeCursor(kThemeArrowCursor);
            gInBackground = false;
          }
          else
            gInBackground = true;
      }
      break;
  }
}

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

void  doMouseDown(EventRecord *eventStrucPtr)
```

```
{
  WindowPartCode partCode;
  WindowRef      windowRef;
  MenuRef        menuRef;
  WindowClass    windowClass;

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

  switch(partCode)
  {
    case inMenuBar:
      menuRef = GetMenuRef(mDemonstration);
      if(gBevelAndImageActive)
        DisableMenuItem(menuRef,iBevelAndImage);
      else
        EnableMenuItem(menuRef,iBevelAndImage);
      doMenuChoice(MenuSelect(eventStrucPtr->where));
      break;

    case inContent:
      GetWindowClass(windowRef,&windowClass);
      if(windowClass == kFloatingWindowClass)
        doMouseDownSmallControls(windowRef,eventStrucPtr);
      else if(windowRef != FrontNonFloatingWindow())
        SelectWindow(windowRef);
      else
      {
        if(gBevelAndImageActive)
          doBevelImagePictIconContent(eventStrucPtr,windowRef);
      }
      break;

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

    case inGoAway:
      if(TrackGoAway(windowRef,eventStrucPtr->where) == true)
      {
        DisposeWindow(windowRef);
        gBevelAndImageActive = false;
      }
      break;

  }
}

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

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;
  MenuRef       menuRef;

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

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        SysBeep(10);
      break;

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

    case mDemonstration:
      switch(menuItem)
      {
        case iBevelAndImage:
          gBevelAndImageActive = true;
          doBevelImagePictIcon();
          break;

        case iTabEditClock:
          doTabEditClock();
          break;

        case iGroupArrowsProgress:
          gGroupArrowsProgressActive = true;
          doGroupArrowsProgress();
          break;

        case iSliders:
          gSlidersActive = true;
          doSliderUserPane();
          break;

        case iTextBoxes:
          doTextBox();
          break;

        case iSmallControls:
          doSmallControls();
          menuRef = GetMenuRef(mDemonstration);
          DisableMenuItem(menuRef,iSmallControls);
          break;
      }
      break;
  }

  HiliteMenu(0);
}

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

void  doUpdate(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;
  RgnHandle visibleRegionHdl = NewRgn();
  Boolean   drawMode = false;

  windowRef = (WindowRef) eventStrucPtr->message;

  BeginUpdate(windowRef);

  SetPortWindowPort(windowRef);

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

  if(gBevelAndImageActive)
  {
    if(windowRef == FrontNonFloatingWindow())
    {
      doDrawMessage(windowRef,!gInBackground);
      doDrawLegends(windowRef,!gInBackground);
    }
    else
    {
      doDrawMessage(windowRef,gInBackground);
      doDrawLegends(windowRef,gInBackground);
    }
  }

  EndUpdate(windowRef);
}

// ********************************************************************************* doActivate

void  doActivate(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;
  Boolean   becomingActive;

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

  doActivateWindow(windowRef,becomingActive);
}

// *************************************************************************** doActivateWindow

void  doActivateWindow(WindowRef windowRef,Boolean becomingActive)
{
  ControlRef controlRef;

  GetRootControl(windowRef,&controlRef);

  if(becomingActive)
  {
    if(gBevelAndImageActive)
    {
      ActivateControl(controlRef);
      doDrawMessage(windowRef,becomingActive);
      doDrawLegends(windowRef,becomingActive);
    }
  }
  else
  {
    if(gBevelAndImageActive)
    {
      DeactivateControl(controlRef);
      doDrawMessage(windowRef,becomingActive);
      doDrawLegends(windowRef,becomingActive);
    }
  }
}

// *************************************************************************** doConcatPStrings

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

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

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

// ******************************************************************************* doCopyPString

void  doCopyPString(Str255 sourceString,Str255 destinationString)
{
  SInt16 stringLength;
  stringLength = sourceString[0];
  BlockMove(sourceString + 1,destinationString + 1,stringLength);
  destinationString[0] = stringLength;
}

// *******************************************************************************************
// BevelImagePictIcon.c
// *******************************************************************************************

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

#include "Controls3.h"

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

WindowRef gWindowRef;

extern Boolean gRunningOnX;
extern Str255  gCurrentString;
extern Boolean gInBackground;

// ********************************************************************** doBevelImagePictIcon

void  doBevelImagePictIcon(void)
{
  BevelDocStrucHandle bevelDocStrucHdl;

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

  doCopyPString("\pBalloon (OS 8/9) and Help tag (OS X) help is available",gCurrentString);

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

  if(!(gWindowRef = GetNewCWindow(rBevelImageWindow,NULL,(WindowRef)-1)))
    ExitToShell();

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

  SetThemeWindowBackground(gWindowRef,kThemeBrushDialogBackgroundActive,true);

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

  if(!(bevelDocStrucHdl = (BevelDocStrucHandle) NewHandle(sizeof(BevelDocStruc))))
    ExitToShell();

  SetWRefCon(gWindowRef,(SInt32) bevelDocStrucHdl);

  // ....................................... get controls, help tags if OS X, and show window

  doGetControls(gWindowRef);

  if(gRunningOnX)
    helpTagsBevelImagePictIcon(gWindowRef);

  ShowWindow(gWindowRef);
}

// ****************************************************************************** doGetControls

void  doGetControls(WindowRef windowRef)
{
  ControlRef                  controlRef;
  BevelDocStrucHandle         bevelDocStrucHdl;
  ThemeButtonKind             buttonKind = kThemeRoundedBevelButton;
  ControlButtonTextPlacement textPlacementBelow = kControlBevelButtonPlaceBelowGraphic;
  ControlButtonTextPlacement textPlacementAbove = kControlBevelButtonPlaceAboveGraphic;
  Boolean                     centrePopupGlyph = true;
  ControlFontStyleRec         controlFontStyleStruc;
```

```
// ..................................... create root control for window, get handle to window's document structure

if(!gRunningOnX)
  CreateRootControl(windowRef,&controlRef);

bevelDocStrucHdl = (BevelDocStrucHandle) (GetWRefCon(windowRef));

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

if(!((*bevelDocStrucHdl)->bevelButton1Ref = GetNewControl(cBevelButton1,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton2Ref = GetNewControl(cBevelButton2,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton3Ref = GetNewControl(cBevelButton3,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton4Ref = GetNewControl(cBevelButton4,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton5Ref = GetNewControl(cBevelButton5,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton6Ref = GetNewControl(cBevelButton6,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton7Ref = GetNewControl(cBevelButton7,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton8Ref = GetNewControl(cBevelButton8,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton9Ref = GetNewControl(cBevelButton9,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton10Ref = GetNewControl(cBevelButton10,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton11Ref = GetNewControl(cBevelButton11,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton12Ref = GetNewControl(cBevelButton12,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton13Ref = GetNewControl(cBevelButton13,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton14Ref = GetNewControl(cBevelButton14,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton15Ref = GetNewControl(cBevelButton15,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton16Ref = GetNewControl(cBevelButton16,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton17Ref = GetNewControl(cBevelButton17,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton18Ref = GetNewControl(cBevelButton18,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton19Ref = GetNewControl(cBevelButton19,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton20Ref = GetNewControl(cBevelButton20,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->bevelButton21Ref = GetNewControl(cBevelButton21,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->imageWell1Ref = GetNewControl(cImageWell1,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->imageWell2Ref = GetNewControl(cImageWell2,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->picture1Ref = GetNewControl(cPicture1,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->picture2Ref = GetNewControl(cPicture2,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->colourIcon1Ref = GetNewControl(cColourIcon1,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->colourIcon2Ref = GetNewControl(cColourIcon2,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->iconSuite1Ref= GetNewControl(cIconSuite1,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->iconSuite2Ref= GetNewControl(cIconSuite2,windowRef)))
  ExitToShell();
if(!((*bevelDocStrucHdl)->windowHeaderRef = GetNewControl(cWindowHeader,windowRef)))
```

```
    ExitToShell();

  // ........................................... if running on OS X, make first four bevel buttons rounded

  if(gRunningOnX)
  {
    SetControlData((*bevelDocStrucHdl)->bevelButton1Ref,kControlEntireControl,
                   kControlBevelButtonKindTag,sizeof(buttonKind),&buttonKind);
    SetControlData((*bevelDocStrucHdl)->bevelButton2Ref,kControlEntireControl,
                   kControlBevelButtonKindTag,sizeof(buttonKind),&buttonKind);
    SetControlData((*bevelDocStrucHdl)->bevelButton3Ref,kControlEntireControl,
                   kControlBevelButtonKindTag,sizeof(buttonKind),&buttonKind);
    SetControlData((*bevelDocStrucHdl)->bevelButton4Ref,kControlEntireControl,
                   kControlBevelButtonKindTag,sizeof(buttonKind),&buttonKind);
  }

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

  SetControlData((*bevelDocStrucHdl)->bevelButton2Ref,kControlEntireControl,
                 kControlBevelButtonTextPlaceTag,sizeof(textPlacementBelow),
                 &textPlacementBelow);
  SetControlData((*bevelDocStrucHdl)->bevelButton21Ref,kControlEntireControl,
                 kControlBevelButtonTextPlaceTag,sizeof(textPlacementAbove),
                 &textPlacementAbove);

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

  SetControlData((*bevelDocStrucHdl)->bevelButton6Ref,kControlEntireControl,
                 kControlBevelButtonCenterPopupGlyphTag,sizeof(centrePopupGlyph),
                 &centrePopupGlyph);

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

  controlFontStyleStruc.flags = kControlUseFontMask;
  controlFontStyleStruc.font = kControlFontSmallBoldSystemFont;
  SetControlFontStyle((*bevelDocStrucHdl)->bevelButton20Ref,&controlFontStyleStruc);

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

  SetControlValue((*bevelDocStrucHdl)->bevelButton3Ref,2);
}

// ************************************************************** doBevelImagePictIconContent

void  doBevelImagePictIconContent(EventRecord *eventStrucPtr,WindowRef windowRef)
{
  BevelDocStrucHandle bevelDocStrucHdl;
  ControlRef          controlRef;
  SInt16              partCode, menuItem;

  bevelDocStrucHdl = (BevelDocStrucHandle) (GetWRefCon(windowRef));

  SetPortWindowPort(windowRef);
  GlobalToLocal(&eventStrucPtr->where);
  partCode = FindControl(eventStrucPtr->where,windowRef,&controlRef);

  doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,1234);

  if(partCode)
  {
    partCode = TrackControl(controlRef,eventStrucPtr->where,NULL);

    if(controlRef == (*bevelDocStrucHdl)->bevelButton1Ref ||
       controlRef == (*bevelDocStrucHdl)->bevelButton2Ref ||
       controlRef == (*bevelDocStrucHdl)->bevelButton3Ref ||
       controlRef == (*bevelDocStrucHdl)->bevelButton4Ref ||
       controlRef == (*bevelDocStrucHdl)->bevelButton9Ref ||
       controlRef == (*bevelDocStrucHdl)->bevelButton13Ref ||
       controlRef == (*bevelDocStrucHdl)->bevelButton14Ref ||
       controlRef == (*bevelDocStrucHdl)->bevelButton15Ref ||
       controlRef == (*bevelDocStrucHdl)->bevelButton16Ref ||
       controlRef == (*bevelDocStrucHdl)->bevelButton17Ref)
    {
      doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,4321);
    }
    else if(controlRef == (*bevelDocStrucHdl)->bevelButton5Ref ||
            controlRef == (*bevelDocStrucHdl)->bevelButton6Ref ||
            controlRef == (*bevelDocStrucHdl)->bevelButton7Ref ||
            controlRef == (*bevelDocStrucHdl)->bevelButton8Ref)
    {
      if(partCode == kControlMenuPart)
      {
        GetBevelButtonMenuValue(controlRef,&menuItem);
        doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,menuItem);
      }
      else
        doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,4321);
    }
    else if(controlRef == (*bevelDocStrucHdl)->bevelButton10Ref ||
            controlRef == (*bevelDocStrucHdl)->bevelButton11Ref ||
            controlRef == (*bevelDocStrucHdl)->bevelButton12Ref)
    {
      if(partCode != kControlEntireControl)
      {
        SetControlValue((*bevelDocStrucHdl)->bevelButton10Ref,0);
        SetControlValue((*bevelDocStrucHdl)->bevelButton11Ref,0);
        SetControlValue((*bevelDocStrucHdl)->bevelButton12Ref,0);
        SetControlValue(controlRef,1);
      }
      doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,4321);
    }
    else if(controlRef == (*bevelDocStrucHdl)->bevelButton18Ref)
    {
      doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,4321);
      doGraphicAlignment(windowRef,controlRef,(*bevelDocStrucHdl)->windowHeaderRef);
    }
    else if(controlRef == (*bevelDocStrucHdl)->bevelButton19Ref)
    {
      doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,4321);
      doTextAlignment(windowRef,controlRef,(*bevelDocStrucHdl)->windowHeaderRef);
    }
    else if(controlRef == (*bevelDocStrucHdl)->bevelButton20Ref)
    {
      doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,4321);
      doTextOffset(windowRef,controlRef,(*bevelDocStrucHdl)->windowHeaderRef);
    }
    else if(controlRef == (*bevelDocStrucHdl)->bevelButton21Ref)
    {
      doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,4321);
      doTextPlacement(windowRef,controlRef,(*bevelDocStrucHdl)->windowHeaderRef);
    }
    else
      doDrawPartCode(windowRef,(*bevelDocStrucHdl)->windowHeaderRef,partCode,4321);
  }
}

// ************************************************************************** doDrawPartCode

void  doDrawPartCode(WindowRef windowRef,ControlRef windowHeaderRef,SInt16 partCode,
                     SInt16 menuItem)
{
  SInt16 stringIndex;
  Str255 theString;

  if(partCode == kControlEntireControl)
    stringIndex = 1;
  else if(partCode == kControlMenuPart)
    stringIndex = 2;
  else if(partCode == kControlTrianglePart)
    stringIndex = 3;
  else if(partCode == kControlEditTextPart)
    stringIndex = 4;
  else if(partCode == kControlPicturePart)
    stringIndex = 5;
  else if(partCode == kControlIconPart)
    stringIndex = 6;
  else if(partCode == kControlClockPart)
    stringIndex = 7;
  else if(partCode == kControlListBoxPart)
    stringIndex = 8;
  else if(partCode == kControlListBoxDoubleClickPart)
    stringIndex = 9;
  else if(partCode == kControlImageWellPart)
    stringIndex = 10;
  else if(partCode == kControlRadioGroupPart)
    stringIndex = 11;
  else if(partCode == kControlButtonPart)
    stringIndex = 12;
  else if(partCode == kControlIndicatorPart)
    stringIndex = 13;

  if(menuItem > 0 && menuItem < 1234)
  {
    doCopyPString("\pTrackControl returned ",gCurrentString);
    GetIndString(theString,rPartCodeStrings,stringIndex);
    doConcatPStrings(gCurrentString,theString);
    doConcatPStrings(gCurrentString,"\p    GetBevelButtonMenuValue returned menu item ");
    NumToString((SInt32) menuItem,theString);
    doConcatPStrings(gCurrentString,theString);
  }
  else if(menuItem == 1234 || menuItem == 4321)
  {
    if(menuItem == 1234)
      doCopyPString("\pMouse-down in ",gCurrentString);
    else if(menuItem == 4321)
      doCopyPString("\pTrackControl returned ",gCurrentString);
    GetIndString(theString,rPartCodeStrings,stringIndex);
    doConcatPStrings(gCurrentString,theString);
  }

  Draw1Control(windowHeaderRef);
  doDrawMessage(windowRef,true);
}

// ********************************************************************** doGraphicAlignment

void  doGraphicAlignment(WindowRef windowRef,ControlRef controlRef,ControlRef windowHeaderRef)
{
  SInt16                        a, b;
  UInt32                        finalTicks;
  ControlButtonGraphicAlignment alignmentConstant = 0;

  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  for(a=1;a<10;a++)
  {
    Delay(60,&finalTicks);
    alignmentConstant++;
    if(alignmentConstant == 9)
      alignmentConstant = 0;

    Draw1Control(windowHeaderRef);
    GetIndString(gCurrentString,rGraphicAlignStrings,a);
    doDrawMessage(windowRef,true);

    for(b=0;b<=52;b++)
    {
      SetBevelButtonGraphicAlignment(controlRef,alignmentConstant,b,b);
      Delay(2,&finalTicks);
      Draw1Control(controlRef);
      QDFlushPortBuffer(GetWindowPort(FrontNonFloatingWindow()),NULL);
    }
  }

  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);
}

// ************************************************************************** doTextAlignment

void  doTextAlignment(WindowRef windowRef,ControlRef controlRef,ControlRef windowHeaderRef)
{
  SInt16                     a, b;
  UInt32                     finalTicks;
  ControlButtonTextAlignment alignmentConstant = -3;

  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  for(a=1;a<4;a++)
  {
    Delay(60,&finalTicks);
    alignmentConstant++;
    if(alignmentConstant == 0)
      alignmentConstant++;

    Draw1Control(windowHeaderRef);
    GetIndString(gCurrentString,rTextAlignStrings,a);
    doDrawMessage(windowRef,true);

    for(b=0;b<=40;b++)
    {
      SetBevelButtonTextAlignment(controlRef,alignmentConstant,b);
      Delay(2,&finalTicks);
      Draw1Control(controlRef);
      QDFlushPortBuffer(GetWindowPort(FrontNonFloatingWindow()),NULL);
    }
  }

  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);
}

// ***************************************************************************** doTextOffset

void  doTextOffset(WindowRef windowRef,ControlRef controlRef,ControlRef windowHeaderRef)
{
  ControlButtonTextAlignment alignmentConstant;
  SInt16                     offset;
  UInt32                     finalTicks;

  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  Draw1Control(windowHeaderRef);
  doCopyPString("\pOffset from left",gCurrentString);
  doDrawMessage(windowRef,true);

  alignmentConstant = kControlBevelButtonAlignTextFlushLeft;
  SetControlData(controlRef,kControlEntireControl,kControlBevelButtonTextAlignTag,
                sizeof(alignmentConstant),&alignmentConstant);
  Draw1Control(controlRef);

  for(offset=1;offset<27;offset++)
  {
    Delay(15,&finalTicks);
    SetControlData(controlRef,kControlEntireControl,kControlBevelButtonTextOffsetTag,
                   sizeof(offset),&offset);
    Draw1Control(controlRef);
    QDFlushPortBuffer(GetWindowPort(FrontNonFloatingWindow()),NULL);
  }

  Delay(60,&finalTicks);

  Draw1Control(windowHeaderRef);
  doCopyPString("\pOffset from right",gCurrentString);
  doDrawMessage(windowRef,true);

  alignmentConstant = kControlBevelButtonAlignTextFlushRight;
  SetControlData(controlRef,kControlEntireControl,kControlBevelButtonTextAlignTag,
                 sizeof(alignmentConstant),&alignmentConstant);

  for(offset=0;offset<13;offset++)
  {
    Delay(15,&finalTicks);
    SetControlData(controlRef,kControlEntireControl,kControlBevelButtonTextOffsetTag,
                   sizeof(offset),&offset);
    Draw1Control(controlRef);
    QDFlushPortBuffer(GetWindowPort(FrontNonFloatingWindow()),NULL);
  }

  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);
}

// ************************************************************************* doTextPlacement

void  doTextPlacement(WindowRef windowRef,ControlRef controlRef,ControlRef windowHeaderRef)
{
  ControlButtonTextPlacement placementConstant;
  UInt32                     finalTicks;

  if(!gRunningOnX)
    SetThemeCursor(kThemeWatchCursor);

  for(placementConstant = 1;placementConstant < 5;placementConstant++)
  {
    Delay(60,&finalTicks);
    SetBevelButtonTextPlacement(controlRef,placementConstant);
    Draw1Control(controlRef);
    Draw1Control(windowHeaderRef);
    GetIndString(gCurrentString,rTextPlacementStrings,placementConstant);
    doDrawMessage(windowRef,true);
    QDFlushPortBuffer(GetWindowPort(FrontNonFloatingWindow()),NULL);
  }

  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);
}

// *************************************************************************** doDrawMessage

void  doDrawMessage(WindowRef windowRef,Boolean inState)
{
  Rect        portRect;
  CFStringRef stringRef;
  Rect        textBoxRect;

  if(windowRef == gWindowRef)
  {
    SetPortWindowPort(windowRef);

    GetWindowPortBounds(windowRef,&portRect);
    stringRef = CFStringCreateWithPascalString(NULL,gCurrentString,kCFStringEncodingMacRoman);
    SetRect(&textBoxRect,portRect.left,7,portRect.right,22);

    if(inState == kThemeStateActive)
      TextMode(srcOr);
    else
      TextMode(grayishTextOr);

    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,inState,true,&textBoxRect,teJustCenter,
                     NULL);
    if(stringRef != NULL)
      CFRelease(stringRef);
  }
}

// ***************************************************************************** doDrawLegends

void  doDrawLegends(WindowRef windowRef,Boolean inState)
{
  Rect theRect;

  if(windowRef == gWindowRef)
  {
    SetPortWindowPort(windowRef);

    if(gRunningOnX)
    {
      SetRect(&theRect,30,54,585,67);
      EraseRect(&theRect);
      OffsetRect(&theRect,0,91);
      EraseRect(&theRect);
      OffsetRect(&theRect,0,92);
      EraseRect(&theRect);
    }

    if(inState == kThemeStateActive)
      TextMode(srcOr);
    else
      TextMode(grayishTextOr);

    SetRect(&theRect,30,53,220,68);
    if(gRunningOnX)
      DrawThemeTextBox(CFSTR("Bevel button content"),kThemeSmallSystemFont,inState,false,
                       &theRect,teJustLeft,NULL);
    else
      DrawThemeTextBox(CFSTR("Bevel sizes and bevel button content"),kThemeSmallSystemFont,
                       inState,false,&theRect,teJustLeft,NULL);
    SetRect(&theRect,313,53,513,68);
    DrawThemeTextBox(CFSTR("Menu position and behaviour"),kThemeSmallSystemFont,inState,
                     false,&theRect,teJustLeft,NULL);
    SetRect(&theRect,30,144,180,159);
    DrawThemeTextBox(CFSTR("Bevel button behaviour"),kThemeSmallSystemFont,inState,false,
                     &theRect,teJustLeft,NULL);
    SetRect(&theRect,313,144,513,159);
    DrawThemeTextBox(CFSTR("Graphic/text alignment & offset"),kThemeSmallSystemFont,inState,
                     false,&theRect,teJustLeft,NULL);
    SetRect(&theRect,490,144,640,159);
    DrawThemeTextBox(CFSTR("Text placement"),kThemeSmallSystemFont,inState,false,&theRect,
                     teJustLeft,NULL);
    SetRect(&theRect,30,235,180,250);
    DrawThemeTextBox(CFSTR("Image wells"),kThemeSmallSystemFont,inState,false,&theRect,
                     teJustLeft,NULL);
    SetRect(&theRect,168,235,308,250);
    DrawThemeTextBox(CFSTR("Picture controls"),kThemeSmallSystemFont,inState,false,&theRect,
                     teJustLeft,NULL);
    SetRect(&theRect,313,235,463,250);
    DrawThemeTextBox(CFSTR("Icon controls (cicn)"),kThemeSmallSystemFont,inState,false,
                     &theRect,teJustLeft,NULL);
    SetRect(&theRect,451,235,601,250);
    DrawThemeTextBox(CFSTR("Icon controls (icon suite)"),kThemeSmallSystemFont,inState,false,
                     &theRect,teJustLeft,NULL);
  }
}

// ************************************************************* helpTagsBevelImagePictIcon

void  helpTagsBevelImagePictIcon(WindowRef windowRef)
{
  BevelDocStrucHandle bevelDocStrucHdl;
  HMHelpContentRec    helpContent;

  bevelDocStrucHdl = (BevelDocStrucHandle) (GetWRefCon(windowRef));

  memset(&helpContent,0,sizeof(helpContent));
  HMSetTagDelay(500);
  HMSetHelpTagsDisplayed(true);

  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideTopCenterAligned;
  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 132;

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 1;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton1Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 2;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton2Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 3;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton3Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 4;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton4Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 5;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton5Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 6;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton6Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 7;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton7Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 8;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton8Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 9;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton9Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 10;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton10Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 10;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton11Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 10;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton12Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 11;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton13Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 11;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton14Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 11;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton15Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 12;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton16Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 13;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton17Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 14;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton18Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 15;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton19Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 16;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton20Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 17;
  HMSetControlHelpContent((*bevelDocStrucHdl)->bevelButton21Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 18;
  HMSetControlHelpContent((*bevelDocStrucHdl)->imageWell1Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 19;
  HMSetControlHelpContent((*bevelDocStrucHdl)->imageWell2Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 20;
  HMSetControlHelpContent((*bevelDocStrucHdl)->picture1Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 21;
  HMSetControlHelpContent((*bevelDocStrucHdl)->picture2Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 22;
  HMSetControlHelpContent((*bevelDocStrucHdl)->colourIcon1Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 23;
  HMSetControlHelpContent((*bevelDocStrucHdl)->colourIcon2Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 24;
  HMSetControlHelpContent((*bevelDocStrucHdl)->iconSuite1Ref,&helpContent);

  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 25;
  HMSetControlHelpContent((*bevelDocStrucHdl)->iconSuite2Ref,&helpContent);
}

// *******************************************************************************************
// TabEditClock.c
// *******************************************************************************************

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

#include "Controls3.h"

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

extern Boolean  gRunningOnX;

// *************************************************************************** doTabEditClock

void  doTabEditClock(void)
{
  DialogRef                     dialogRef;
  ControlRef                    controlRef;
  Str255                        initialName = "\pYour name here";
  ControlEditTextSelectionRec   selectionRec;
  ControlKeyFilterUPP           numericFilterUPP;
  ModalFilterUPP                eventFilterUPP;
  ControlEditTextValidationUPP  editTextValidatorUPP;
  SInt16                        tabHit, itemHit;

  if(FrontNonFloatingWindow())
    doActivateWindow(FrontNonFloatingWindow(),false);
  if(!(dialogRef = GetNewDialog(rTabEditClockDialog,NULL,(WindowRef) -1)))
    ExitToShell();

  SetWTitle(GetDialogWindow(dialogRef),"\pTab, Edit Text, and Clock Controls");
  SetPortDialogPort(dialogRef);

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

  SetDialogDefaultItem(dialogRef,kStdOkItemIndex);
  SetDialogTracksCursor(dialogRef,true);

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

  GetDialogItemAsControl(dialogRef,iClocksUserPane,&controlRef);
  HideControl(controlRef);

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

  GetDialogItemAsControl(dialogRef,iEditText1,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlEditTextTextTag,initialName[0],
               &initialName[1]);
  selectionRec.selStart = 0;
  selectionRec.selEnd = 32767;
  SetControlData(controlRef,kControlEntireControl,kControlEditTextSelectionTag,
               sizeof(selectionRec),&selectionRec);

  // create universal procedure pointers for event filter, key filter, and edit text validator

  eventFilterUPP        = NewModalFilterUPP((ModalFilterProcPtr) eventFilter);
  numericFilterUPP      = NewControlKeyFilterUPP((ControlKeyFilterProcPtr) numericFilter);
  editTextValidatorUPP  = NewControlEditTextValidationUPP((ControlEditTextValidationProcPtr)
                                                          editTextValidator);

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

  GetDialogItemAsControl(dialogRef,iEditText1,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlEditTextValidationProcTag,
               sizeof(editTextValidatorUPP),&editTextValidatorUPP);

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

  GetDialogItemAsControl(dialogRef,iEditText2,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlEditTextKeyFilterTag,
               sizeof(numericFilterUPP),&numericFilterUPP);

  // ................................ set help tags, show dialog, and enter modal dialog loop

  if(gRunningOnX)
    helpTagsTabEditClock(dialogRef);

  ShowWindow(GetDialogWindow(dialogRef));

  do
  {
    ModalDialog(eventFilterUPP,&itemHit);

    if(itemHit == iTabs)
    {
      GetDialogItemAsControl(dialogRef,iTabs,&controlRef);
      tabHit = GetControlValue(controlRef);

      if(tabHit == tabEditText)
      {
        SetWTitle(GetDialogWindow(dialogRef),"\pTab, Edit Text, and Clock Controls");

        GetDialogItemAsControl(dialogRef,iClocksUserPane,&controlRef);
        HideControl(controlRef);
        GetDialogItemAsControl(dialogRef,iEditTextUserPane,&controlRef);
        ActivateControl(controlRef);
        ShowControl(controlRef);

        GetDialogItemAsControl(dialogRef,iEditText1,&controlRef);
        SetKeyboardFocus(GetDialogWindow(dialogRef),controlRef,kControlFocusNextPart);
      }
      else if(tabHit == tabClocks)
      {
        SetWTitle(GetDialogWindow(dialogRef),"\pTab, Clock, and Edit Text Controls");

        GetDialogItemAsControl(dialogRef,iEditTextUserPane,&controlRef);
        DeactivateControl(controlRef);
        HideControl(controlRef);

        GetDialogItemAsControl(dialogRef,iClocksUserPane,&controlRef);
        ShowControl(controlRef);

        GetDialogItemAsControl(dialogRef,iClocks1,&controlRef);
        SetKeyboardFocus(GetDialogWindow(dialogRef),controlRef,kControlFocusNextPart);
      }
    }
    else if(itemHit == iExtractEditText)
    {
      GetDialogItemAsControl(dialogRef,iImageWellEditText,&controlRef);
      Draw1Control(controlRef);
      doExtractEditText(dialogRef);
    }
    else if(itemHit == iExtractClocks)
    {
      GetDialogItemAsControl(dialogRef,iImageWellClocks,&controlRef);
      Draw1Control(controlRef);
      doExtractDateTime(dialogRef);
    }
  } while(itemHit != kStdOkItemIndex);

  DisposeDialog(dialogRef);
  DisposeModalFilterUPP(eventFilterUPP);
  DisposeControlKeyFilterUPP(numericFilterUPP);
  DisposeControlEditTextValidationUPP(editTextValidatorUPP);
}

// *********************************************************************** doExtractEditText

void  doExtractEditText(DialogRef dialogRef)
{
  GrafPtr     oldPort;
  RGBColor    saveBackColour, whiteColour = { 0xFFFF, 0xFFFF, 0xFFFF };
  SInt16      iType;
  Handle      theHandle;
  Rect        theRect;
  Str255      theString;
  CFStringRef stringRef;
  ControlRef  controlRef;
  Size        actualSize;

  GetPort(&oldPort);
  SetPortDialogPort(dialogRef);
  GetBackColor(&saveBackColour);
  RGBBackColor(&whiteColour);

  GetDialogItem(dialogRef,iEditText1,&iType,&theHandle,&theRect);
  GetDialogItemText(theHandle,theString);
  if(!gRunningOnX)
  {
    MoveTo(124,177);
    DrawString(theString);
  }
  else
  {
    stringRef = CFStringCreateWithPascalString(NULL,theString,kCFStringEncodingMacRoman);
    SetRect(&theRect,124,166,314,181);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
  }

  GetDialogItem(dialogRef,iEditText2,&iType,&theHandle,&theRect);
  GetDialogItemText(theHandle,theString);
  if(!gRunningOnX)
  {
    MoveTo(124,190);
    DrawString(theString);
  }
  else
  {
    stringRef = CFStringCreateWithPascalString(NULL,theString,kCFStringEncodingMacRoman);
    SetRect(&theRect,124,179,314,194);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
  }

  GetDialogItemAsControl(dialogRef,iEditText3,&controlRef);
  GetControlDataSize(controlRef,kControlEditTextPart,kControlEditTextTextTag,&actualSize);
  GetControlData(controlRef,kControlEntireControl,kControlEditTextPasswordTag,actualSize,
                 &theString[1],NULL);
  theString[0] = actualSize;
  if(!gRunningOnX)
  {
    MoveTo(124,203);
    DrawString(theString);
  }
  else
  {
    stringRef = CFStringCreateWithPascalString(NULL,theString,kCFStringEncodingMacRoman);
    SetRect(&theRect,124,192,314,207);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
  }

  if(gRunningOnX)
  {
    if(stringRef != NULL)
      CFRelease(stringRef);
  }

  RGBBackColor(&saveBackColour);
  SetPort(oldPort);
}

// *********************************************************************** doExtractDateTime

void  doExtractDateTime(DialogRef dialogRef)
{
  GrafPtr     oldPort;
  RGBColor    saveBackColour, whiteColour = { 0xFFFF, 0xFFFF, 0xFFFF };
  ControlRef  controlRef;
  LongDateRec longDateTimeStruc;
  SInt16      second, minute, hour, day, month, year;
  Str255      theString, tempString;
  CFStringRef stringRef;
  Rect        theRect;

  GetPort(&oldPort);
  SetPortDialogPort(dialogRef);
  GetBackColor(&saveBackColour);
  RGBBackColor(&whiteColour);

  GetDialogItemAsControl(dialogRef,iClocks1,&controlRef);
  GetControlData(controlRef,kControlEntireControl,kControlClockLongDateTag,
                 sizeof(longDateTimeStruc),&longDateTimeStruc,NULL);
  hour = longDateTimeStruc.ld.hour;
```

```
minute = longDateTimeStruc.ld.minute;
second = longDateTimeStruc.ld.second;
doCopyPString("\pHour ",theString);
NumToString((SInt32) hour,tempString);
doConcatPStrings(theString,tempString);
doConcatPStrings(theString,"\p, Minute ");
NumToString((SInt32) minute,tempString);
doConcatPStrings(theString,tempString);
doConcatPStrings(theString,"\p, Second ");
NumToString((SInt32) second,tempString);
doConcatPStrings(theString,tempString);
MoveTo(124,177);
if(!gRunningOnX)
  DrawString(theString);
else
{
  stringRef = CFStringCreateWithPascalString(NULL,theString,kCFStringEncodingMacRoman);
  SetRect(&theRect,124,166,314,181);
  DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
  if(stringRef != NULL)
    CFRelease(stringRef);
}

GetDialogItemAsControl(dialogRef,iClocks2,&controlRef);
GetControlData(controlRef,kControlEntireControl,kControlClockLongDateTag,
               sizeof(longDateTimeStruc),&longDateTimeStruc,NULL);
day = longDateTimeStruc.ld.day;
month = longDateTimeStruc.ld.month;
year = longDateTimeStruc.ld.year;
doCopyPString("\pDay ",theString);
NumToString((SInt32) day,tempString);
doConcatPStrings(theString,tempString);
doConcatPStrings(theString,"\p, Month ");
NumToString((SInt32) month,tempString);
doConcatPStrings(theString,tempString);
doConcatPStrings(theString,"\p, Year ");
NumToString((SInt32) year,tempString);
doConcatPStrings(theString,tempString);
MoveTo(124,190);
if(!gRunningOnX)
  DrawString(theString);
else
{
  stringRef = CFStringCreateWithPascalString(NULL,theString,kCFStringEncodingMacRoman);
  SetRect(&theRect,124,179,314,194);
  DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
  if(stringRef != NULL)
    CFRelease(stringRef);
}

GetDialogItemAsControl(dialogRef,iClocks3,&controlRef);
GetControlData(controlRef,kControlEntireControl,kControlClockLongDateTag,
               sizeof(longDateTimeStruc),&longDateTimeStruc,NULL);
month = longDateTimeStruc.ld.month;
year = longDateTimeStruc.ld.year;
doCopyPString("\pMonth ",theString);
NumToString((SInt32) month,tempString);
doConcatPStrings(theString,tempString);
doConcatPStrings(theString,"\p, Year ");
NumToString((SInt32) year,tempString);
doConcatPStrings(theString,tempString);
MoveTo(124,203);
if(!gRunningOnX)
  DrawString(theString);
else
{
  stringRef = CFStringCreateWithPascalString(NULL,theString,kCFStringEncodingMacRoman);
  SetRect(&theRect,124,192,314,207);
  DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
```

```
    if(stringRef != NULL)
      CFRelease(stringRef);
  }

  RGBBackColor(&saveBackColour);
  SetPort(oldPort);
}

// *************************************************************************** numericFilter

ControlKeyFilterResult  numericFilter(ControlRef controlRef,SInt16* keyCode,SInt16 *charCode,
                                      EventModifiers *modifiers)
{
  if(((char) *charCode >= '0') && ((char) *charCode <= '9') ||
     (BitTst(modifiers,15 - cmdKeyBit)))
  {
    return kControlKeyFilterPassKey;
  }

  switch(*charCode)
  {
    case kLeftArrow:
    case kRightArrow:
    case kUpArrow:
    case kDownArrow:
    case kBackspace:
    case kDelete:
      return kControlKeyFilterPassKey;
      break;
  }

  SysBeep(10);
  return kControlKeyFilterBlockKey;
}

// *********************************************************************** editTextValidator

void  editTextValidator(ControlRef controlRef)
{
  Str255 oldText, newText;
  Size   actualSize;
  UInt8  a, count = 0;

  GetControlData(controlRef,kControlEntireControl,kControlEditTextTextTag,sizeof(oldText) -1,
                 &oldText[1],&actualSize);

  if(actualSize <= 255)
    oldText[0] = actualSize;
  else
    oldText[0] = 255;

  for(a=1;a<=oldText[0];a++)
  {
    if(((oldText[a] >= 'A') && (oldText[a] <= 'Z') ||
        (oldText[a] >= 'a') && (oldText[a] <= 'z')) ||
        (oldText[a] == ' ') || (oldText[a] == '.'))
    {
      newText[count + 1] = oldText[a];
      count++;
    }
  }

  newText[0] = count;

  SetControlData(controlRef,kControlEntireControl,kControlEditTextTextTag,newText[0],
                 &newText[1]);

  Draw1Control(controlRef);
}
```

```
// ********************************************************************* helpTagsTabEditClock

void  helpTagsTabEditClock(DialogRef dialogRef)
{
  HMHelpContentRec helpContent;
  SInt16           a;
  static SInt16    itemNumber[10] = { 2,5,7,9,11,21,14,16,18,20 };
  ControlRef       controlRef;

  memset(&helpContent,0,sizeof(helpContent));
  HMSetTagDelay(500);
  HMSetHelpTagsDisplayed(true);

  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideTopCenterAligned;
  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 133;

  for(a = 1;a <= 10; a++)
  {
    helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = a;
    GetDialogItemAsControl(dialogRef,itemNumber[a - 1],&controlRef);
    HMSetControlHelpContent(controlRef,&helpContent);
  }
}

// *******************************************************************************************
// GroupArrowsProgress.c
// *******************************************************************************************

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

#include "Controls3.h"

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

ControlRef       gCacheSizeControlRef;
ControlRef       gLittleArrowsControlRef;
ControlActionUPP gArrowsActionFunctionUPP;
extern Boolean   gRunningOnX;
extern Boolean   gGroupArrowsProgressActive;

// ******************************************************************** doGroupArrowsProgress

void  doGroupArrowsProgress(void)
{
  DialogRef      dialogRef;
  ControlRef     controlRef;
  ModalFilterUPP eventFilterUPP;
  SInt16         controlValue, itemHit;
  Str255         theString;

  if(FrontNonFloatingWindow())
    doActivateWindow(FrontNonFloatingWindow(),false);

  if(!(dialogRef = GetNewDialog(rGroupArrowsProgDialog,NULL,(WindowRef) -1)))
    ExitToShell();

  SetPortDialogPort(dialogRef);

  // ........................................................... set default button, help tags

  SetDialogDefaultItem(dialogRef,kStdOkItemIndex);

  // ...... create universal procedure pointers for event filter and little arrows action function

  eventFilterUPP            = NewModalFilterUPP((ModalFilterProcPtr) eventFilter);
  gArrowsActionFunctionUPP  = NewControlActionUPP((ControlActionProcPtr) arrowsActionFunction);
```

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

GetDialogItemAsControl(dialogRef,iCheckboxGroup,&controlRef);
SetControlValue(controlRef,1);

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

GetDialogItemAsControl(dialogRef,iLittleArrows,&gLittleArrowsControlRef);
NumToString((SInt32) GetControlValue(gLittleArrowsControlRef),theString);
doConcatPStrings(theString,"\pK");
GetDialogItemAsControl(dialogRef,iStaticTextCache,&gCacheSizeControlRef);
SetControlData(gCacheSizeControlRef,kControlEntireControl,kControlStaticTextTextTag,
               theString[0],&theString[1]);

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

GetDialogItemAsControl(dialogRef,iUserPaneScoreAverage,&controlRef);
HideControl(controlRef);

// ...................................... set help tags, show dialog, and enter ModalDialog loop

if(gRunningOnX)
  helpTagsGroupArrowsProgress(dialogRef);

ShowWindow(GetDialogWindow(dialogRef));

do
{
  ModalDialog(eventFilterUPP,&itemHit);

  if(itemHit == iCheckboxGroup)
  {
    doCheckBoxGroupBox(dialogRef);
  }
  else if(itemHit == iPopupGroup)
  {
    doPopupGroupBox(dialogRef);
  }
  else if(itemHit == iCheckboxShowInitials)
  {
    GetDialogItemAsControl(dialogRef,iCheckboxShowInitials,&controlRef);
    controlValue = (!(GetControlValue(controlRef)));
    SetControlValue(controlRef,controlValue);
  }
  else if(itemHit == iCheckboxShowAverages)
  {
    GetDialogItemAsControl(dialogRef,iCheckboxShowAverages,&controlRef);
    controlValue = (!(GetControlValue(controlRef)));
    SetControlValue(controlRef,controlValue);
  }
  else if(itemHit == iDisclosureTriangle)
  {
    doChasingAndProgress(dialogRef);
  }
  else if(itemHit == iPushButtonExtract)
  {
    GetDialogItemAsControl(dialogRef,iImageWell,&controlRef);
    Draw1Control(controlRef);
    doExtractCurrentStatus(dialogRef);
  }
} while(itemHit != kStdOkItemIndex);

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

DisposeDialog(dialogRef);
DisposeModalFilterUPP(eventFilterUPP);
DisposeControlActionUPP(gArrowsActionFunctionUPP);
```

```
  gGroupArrowsProgressActive = false;
}

// ********************************************************************** doCheckBoxGroupBox

void  doCheckBoxGroupBox(DialogRef dialogRef)
{
  ControlRef controlRef;
  SInt16     controlValue;

  GetDialogItemAsControl(dialogRef,iCheckboxGroup,&controlRef);
  controlValue = (!(GetControlValue(controlRef)));
  SetControlValue(controlRef,controlValue);

  if(controlValue == 0)
  {
    GetDialogItemAsControl(dialogRef,iRadioGroupColour,&controlRef);
    DeactivateControl(controlRef);
    GetDialogItemAsControl(dialogRef,iStaticTextColourDepth,&controlRef);
    DeactivateControl(controlRef);
  }
  else if(controlValue == 1)
  {
    GetDialogItemAsControl(dialogRef,iRadioGroupColour,&controlRef);
    ActivateControl(controlRef);
    GetDialogItemAsControl(dialogRef,iStaticTextColourDepth,&controlRef);
    ActivateControl(controlRef);
  }
}

// ************************************************************************ doPopupGroupBox

void  doPopupGroupBox(DialogRef dialogRef)
{
  ControlRef controlRef;
  SInt16     controlValue;

  GetDialogItemAsControl(dialogRef,iPopupGroup,&controlRef);
  controlValue = GetControlValue(controlRef);

  if(controlValue == 1)
  {
    GetDialogItemAsControl(dialogRef,iUserPaneScoreAverage,&controlRef);
    HideControl(controlRef);
    GetDialogItemAsControl(dialogRef,iUserPaneNamesInitials,&controlRef);
    ShowControl(controlRef);
  }
  else if(controlValue == 2)
  {
    GetDialogItemAsControl(dialogRef,iUserPaneNamesInitials,&controlRef);
    HideControl(controlRef);
    GetDialogItemAsControl(dialogRef,iUserPaneScoreAverage,&controlRef);
    ShowControl(controlRef);
  }
}

// ******************************************************************** doChasingAndProgress

void  doChasingAndProgress(DialogRef dialogRef)
{
  ControlRef controlRef;
  SInt16     controlValue;
  Handle     ditlHdl;
  Boolean    indeterminateFlag = 1;
  Str255     expandString   = "\pHide Progress Bar and Chasing Arrows";
  Str255     collapseString = "\pShow Progress Bar and Chasing Arrows";

  GetDialogItemAsControl(dialogRef,iDisclosureTriangle,&controlRef);
  controlValue = (!(GetControlValue(controlRef)));
  SetControlValue(controlRef,controlValue);

  if(controlValue == 1)
  {
    ditlHdl = GetResource('DITL',130);
    AppendDITL(dialogRef,ditlHdl,appendDITLBottom);
    ReleaseResource(ditlHdl);

    GetDialogItemAsControl(dialogRef,iProgressBar,&controlRef);
    SetControlData(controlRef,kControlEntireControl,kControlProgressBarIndeterminateTag,
                   sizeof(indeterminateFlag),&indeterminateFlag);

    GetDialogItemAsControl(dialogRef,kStdOkItemIndex,&controlRef);
    MoveControl(controlRef,277,351);

    GetDialogItemAsControl(dialogRef,iStaticTextDisclosure,&controlRef);
    SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,
                   expandString[0],&expandString[1]);
    Draw1Control(controlRef);
  }
  else if(controlValue == 0)
  {
    GetDialogItemAsControl(dialogRef,kStdOkItemIndex,&controlRef);
    MoveControl(controlRef,277,280);

    ShortenDITL(dialogRef,3);
    SizeWindow(GetDialogWindow(dialogRef),362,321,false);

    GetDialogItemAsControl(dialogRef,iStaticTextDisclosure,&controlRef);
    SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,
                   collapseString[0],&collapseString[1]);
    Draw1Control(controlRef);
  }
}

// ****************************************************************** doExtractCurrentStatus

void  doExtractCurrentStatus(DialogRef dialogRef)
{
  GrafPtr     oldPort;
  RGBColor    saveBackColour, whiteColour = { 0xFFFF, 0xFFFF, 0xFFFF };
  ControlRef  controlRef;
  SInt16      controlValue;
  Str255      theString;
  CFStringRef stringRef;
  Rect        theRect;

  GetPort(&oldPort);
  SetPortDialogPort(dialogRef);
  GetBackColor(&saveBackColour);
  RGBBackColor(&whiteColour);

  GetDialogItemAsControl(dialogRef,iCheckboxGroup,&controlRef);
  controlValue = GetControlValue(controlRef);
  if(controlValue)
  {
    doCopyPString("\pUse colour,",theString);
    GetDialogItemAsControl(dialogRef,iRadioGroupColour,&controlRef);
    controlValue = GetControlValue(controlRef);
    if(controlValue == 1)
      doConcatPStrings(theString,"\p 8 bit.");
    else if(controlValue == 2)
      doConcatPStrings(theString,"\p 16 bit.");
    else if(controlValue == 3)
      doConcatPStrings(theString,"\p 32 bit.");
  }
  else
    doCopyPString("\pDont use colour.",theString);
```

```
if(!gRunningOnX)
{
  MoveTo(108,216);
  DrawString(theString);
}
else
{
  stringRef = CFStringCreateWithPascalString(NULL,theString,kCFStringEncodingMacRoman);
  SetRect(&theRect,108,205,347,220);
  DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
  if(stringRef != NULL)
    CFRelease(stringRef);
}

GetDialogItemAsControl(dialogRef,iPopupGroup,&controlRef);
controlValue = GetControlValue(controlRef);
if(controlValue == 1)
{
  doCopyPString("\pPlayer, ",theString);
  GetDialogItemAsControl(dialogRef,iRadioGroupNames,&controlRef);
  controlValue = GetControlValue(controlRef);
  if(controlValue == 1)
    doConcatPStrings(theString,"\pname first,");
  else if(controlValue == 2)
    doConcatPStrings(theString,"\pname last,");
  GetDialogItemAsControl(dialogRef,iCheckboxShowInitials,&controlRef);
  controlValue = GetControlValue(controlRef);
  if(controlValue == 1)
    doConcatPStrings(theString,"\p show number.");
  else if(controlValue == 0)
    doConcatPStrings(theString,"\p no number.");
}
else if(controlValue == 2)
{
  doCopyPString("\pScore, ",theString);
  GetDialogItemAsControl(dialogRef,iRadioGroupScores,&controlRef);
  controlValue = GetControlValue(controlRef);
  if(controlValue == 1)
    doConcatPStrings(theString,"\pbatting, ");
  else if(controlValue == 2)
    doConcatPStrings(theString,"\pbowling, ");
  GetDialogItemAsControl(dialogRef,iCheckboxShowAverages,&controlRef);
  controlValue = GetControlValue(controlRef);
  if(controlValue == 1)
    doConcatPStrings(theString,"\pshow average.");
  else if(controlValue == 0)
    doConcatPStrings(theString,"\pno average.");
}

if(!gRunningOnX)
{
  MoveTo(108,229);
  DrawString(theString);
}
else
{
  stringRef = CFStringCreateWithPascalString(NULL,theString,kCFStringEncodingMacRoman);
  SetRect(&theRect,108,219,347,234);
  DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
  if(stringRef != NULL)
    CFRelease(stringRef);
}

if(!gRunningOnX)
{
  MoveTo(108,242);
  DrawString("\pCache size: ");
}
else
```

```
  {
    stringRef = CFStringCreateWithPascalString(NULL,"\pCache size: ",
                                           kCFStringEncodingMacRoman);
    SetRect(&theRect,108,232,347,247);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
    if(stringRef != NULL)
      CFRelease(stringRef);
  }

  GetDialogItemAsControl(dialogRef,iLittleArrows,&controlRef);
  NumToString((SInt32) GetControlValue(controlRef),theString);
  if(!gRunningOnX)
    DrawString(theString);
  else
  {
    stringRef = CFStringCreateWithPascalString(NULL,theString,kCFStringEncodingMacRoman);
    SetRect(&theRect,174,232,347,247);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,true,false,&theRect,teJustLeft,NULL);
    if(stringRef != NULL)
      CFRelease(stringRef);
  }

  RGBBackColor(&saveBackColour);
  SetPort(oldPort);
}

// ************************************************************** helpTagsGroupArrowsProgress

void  helpTagsGroupArrowsProgress(DialogRef dialogRef)
{
  HMHelpContentRec helpContent;
  SInt16           a;
  static SInt16    itemNumber[8] = { 2,9,21,25,27,29,31,32 };
  ControlRef       controlRef;

  memset(&helpContent,0,sizeof(helpContent));
  HMSetTagDelay(500);
  HMSetHelpTagsDisplayed(true);

  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideTopCenterAligned;
  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 134;

  for(a = 1;a <= 8; a++)
  {
    helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = a;
    GetDialogItemAsControl(dialogRef,itemNumber[a - 1],&controlRef);
    HMSetControlHelpContent(controlRef,&helpContent);
  }
}

// *********************************************************************************************
// Sliders.c
// *********************************************************************************************

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

#include "Controls3.h"

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

extern Boolean gRunningOnX;

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

ControlActionUPP           gSliderActionFunction1UPP;
ControlActionUPP           gSliderActionFunction2UPP;
ControlUserPaneDrawUPP     gUserPaneDrawFunctionUPP;
```

```
ControlUserPaneActivateUPP gUserPaneActivateFunctionUPP;
ControlRef                 gSlider1Ref;
ControlRef                 gSlider2Ref;
ControlRef                 gSlider3Ref;
ControlRef                 gSlider4Ref;
ControlRef                 gSlider5Ref;
ControlRef                 gSlider6Ref;
RGBColor                   gRedColour;
RGBColor                   gBlueColour;
RGBColor                   gBlackColour   = { 0x0000, 0x0000, 0x0000 };
Boolean                    gDrawActivated = true;
extern Boolean             gSlidersActive;

// ************************************************************************ doSliderUserPane

void  doSliderUserPane(void)
{
  DialogRef      dialogRef;
  ModalFilterUPP eventFilterUPP;
  ControlRef     controlRef;
  SInt16         itemHit;

  if(FrontNonFloatingWindow())
    doActivateWindow(FrontNonFloatingWindow(),false);

  if(!(dialogRef = GetNewDialog(rSlidersDialog,NULL,(WindowRef) -1)))
    ExitToShell();

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

  SetDialogDefaultItem(dialogRef,kStdOkItemIndex);

  // ........................................ create universal procedure pointer for event filter function

  eventFilterUPP = NewModalFilterUPP((ModalFilterProcPtr) eventFilter);

  // ...................................... create universal procedure pointers for slider action functions

  gSliderActionFunction1UPP = NewControlActionUPP((ControlActionProcPtr)
                              sliderActionFunction1);
  gSliderActionFunction2UPP = NewControlActionUPP((ControlActionProcPtr)
                              sliderActionFunction2);

  // ........ create universal procedure pointers for user pane functions, set user pane functions

  gUserPaneDrawFunctionUPP = NewControlUserPaneDrawUPP((ControlUserPaneDrawProcPtr)
                                                       userPaneDrawFunction);
  GetDialogItemAsControl(dialogRef,iUserPane1,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlUserPaneDrawProcTag,
                 sizeof(gUserPaneDrawFunctionUPP),&gUserPaneDrawFunctionUPP);

  gUserPaneActivateFunctionUPP = NewControlUserPaneActivateUPP((ControlUserPaneActivateProcPtr)
                                                               userPaneActivateFunction);
  GetDialogItemAsControl(dialogRef,iUserPane1,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlUserPaneActivateProcTag,
                 sizeof(gUserPaneActivateFunctionUPP),&gUserPaneActivateFunctionUPP);

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

  GetDialogItemAsControl(dialogRef,iSlider1,&gSlider1Ref);
  doDrawSliderValues(dialogRef,gSlider1Ref);
  GetDialogItemAsControl(dialogRef,iSlider2,&gSlider2Ref);
  doDrawSliderValues(dialogRef,gSlider2Ref);
  GetDialogItemAsControl(dialogRef,iSlider3,&gSlider3Ref);
  doDrawSliderValues(dialogRef,gSlider3Ref);
  GetDialogItemAsControl(dialogRef,iSlider4,&gSlider4Ref);
  doDrawSliderValues(dialogRef,gSlider4Ref);

  // ................................. get control handles and values for bottom two sliders, set colour
  GetDialogItemAsControl(dialogRef,iSlider5,&gSlider5Ref);
  gRedColour.red = 2 * GetControlValue(gSlider5Ref);
  GetDialogItemAsControl(dialogRef,iSlider6,&gSlider6Ref);
  gBlueColour.blue = 2 * GetControlValue(gSlider5Ref);

  // ................................... set help tags, show dialog, and enter ModalDialog loop

  if(gRunningOnX)
    helpTagsSliders(dialogRef);

  ShowWindow(GetDialogWindow(dialogRef));

  do
  {
    ModalDialog(eventFilterUPP,&itemHit);
  } while(itemHit != kStdOkItemIndex);

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

  DisposeDialog(dialogRef);
  DisposeModalFilterUPP(eventFilterUPP);
  DisposeControlActionUPP(gSliderActionFunction1UPP);
  DisposeControlActionUPP(gSliderActionFunction2UPP);
  DisposeControlUserPaneDrawUPP(gUserPaneDrawFunctionUPP);
  DisposeControlUserPaneActivateUPP(gUserPaneActivateFunctionUPP);
  gSlidersActive = false;
}

// *********************************************************************** doDrawSliderValues

void  doDrawSliderValues(DialogRef dialogRef,ControlRef controlRef)
{
  Str255  theString;
  SInt16  staticTextItem;

  NumToString((SInt32) GetControlValue(controlRef),theString);

  if(controlRef == gSlider1Ref)
    staticTextItem = iSlider1StaticText;
  else if(controlRef == gSlider2Ref)
    staticTextItem = iSlider2StaticText;
  else if(controlRef == gSlider3Ref)
    staticTextItem = iSlider3StaticText;
  else if(controlRef == gSlider4Ref)
    staticTextItem = iSlider4StaticText;

  GetDialogItemAsControl(dialogRef,staticTextItem,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,theString[0],
                 &theString[1]);
  Draw1Control(controlRef);
}

// ********************************************************************* userPaneDrawFunction

void  userPaneDrawFunction(ControlRef theControl,SInt16 thePart)
{
  Rect theRect;

  SetRect(&theRect,218,175,238,195);
  DrawThemePlacard(&theRect,gDrawActivated);
  InsetRect(&theRect,2,2);

  if(gDrawActivated)
  {
    RGBForeColor(&gRedColour);
    PaintRect(&theRect);
  }
  SetRect(&theRect,218,196,238,216);
  DrawThemePlacard(&theRect,gDrawActivated);
  InsetRect(&theRect,2,2);

  if(gDrawActivated)
  {
    RGBForeColor(&gBlueColour);
    PaintRect(&theRect);
  }

 }

// ******************************************************************* userPaneActivateFunction

void  userPaneActivateFunction(ControlRef control,Boolean activating)
{
  if(activating)
    gDrawActivated = true;
  else
    gDrawActivated = false;
}

// **************************************************************************** helpTagsSliders

void  helpTagsSliders(DialogRef dialogRef)
{
  HMHelpContentRec helpContent;
  SInt16           a;
  static SInt16    itemNumber[7] = { 2,3,4,5,17,19,18 };
  ControlRef       controlRef;

  memset(&helpContent,0,sizeof(helpContent));
  HMSetTagDelay(500);
  HMSetHelpTagsDisplayed(true);

  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideTopCenterAligned;
  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 135;

  for(a = 1;a <= 7; a++)
  {
    helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = a;
    GetDialogItemAsControl(dialogRef,itemNumber[a - 1],&controlRef);
    HMSetControlHelpContent(controlRef,&helpContent);
  }
}

// *******************************************************************************************
// TextBox.c
//*******************************************************************************************

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

#include "Controls3.h"

// ********************************************************************************* doTextBox

void  doTextBox(void)
{
  DialogRef      dialogRef;
  ModalFilterUPP eventFilterUPP;
  SInt16         itemHit;

  if(FrontNonFloatingWindow())
    doActivateWindow(FrontNonFloatingWindow(),false);

  if(!(dialogRef = GetNewDialog(rAboutDialog,NULL,(WindowRef) -1)))
    ExitToShell();
  // .............................................................................. set default button

  SetDialogDefaultItem(dialogRef,kStdOkItemIndex);

  // ................................................ create universal procedure pointer for event filter

  eventFilterUPP = NewModalFilterUPP((ModalFilterProcPtr) eventFilter);

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

  ShowWindow(GetDialogWindow(dialogRef));

  do
  {
    ModalDialog(eventFilterUPP,&itemHit);
  } while(itemHit != kStdOkItemIndex);

  DisposeDialog(dialogRef);
  DisposeModalFilterUPP(eventFilterUPP);
}

// *******************************************************************************************
// Callbacks.c
//*******************************************************************************************

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

#include "Controls3.h"

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

extern Boolean           gGroupArrowsProgressActive;
extern Boolean           gSlidersActive;
extern ControlActionUPP gArrowsActionFunctionUPP;
extern ControlActionUPP gSliderActionFunction1UPP;
extern ControlActionUPP gSliderActionFunction2UPP;
extern ControlRef        gLittleArrowsControlRef;
extern ControlRef        gCacheSizeControlRef;
extern ControlRef        gSlider1Ref;
extern ControlRef        gSlider2Ref;
extern ControlRef        gSlider3Ref;
extern ControlRef        gSlider4Ref;
extern ControlRef        gSlider5Ref;
extern ControlRef        gSlider6Ref;
extern RGBColor          gRedColour;
extern RGBColor          gBlueColour;
extern RGBColor          gBlackColour;

// ******************************************************************************** eventFilter

Boolean  eventFilter(DialogRef dialogRef,EventRecord *eventStrucPtr,SInt16 *itemHit)
{
  Boolean    handledEvent;
  GrafPtr    oldPort;
  Point      mouseXY;
  ControlRef controlRef;

  handledEvent = false;

  if((eventStrucPtr->what == updateEvt) &&
     ((WindowRef) eventStrucPtr->message != GetDialogWindow(dialogRef)))
  {
    doUpdate(eventStrucPtr);
  }
  else if((eventStrucPtr->what == autoKey) && ((eventStrucPtr->modifiers & cmdKey) != 0))
  {
    handledEvent = true;
    return handledEvent;
  }
  else
  {
    GetPort(&oldPort);
    SetPortDialogPort(dialogRef);

    if(gGroupArrowsProgressActive)
    {
      if(eventStrucPtr->what == mouseDown)
      {
        mouseXY = eventStrucPtr->where;
        GlobalToLocal(&mouseXY);
        if(FindControl(mouseXY,GetDialogWindow(dialogRef),&controlRef))
        {
          if(controlRef == gLittleArrowsControlRef)
          {
            TrackControl(controlRef,mouseXY,gArrowsActionFunctionUPP);
            handledEvent = true;
          }
        }
      }
    }
    else if(gSlidersActive)
    {
      if(eventStrucPtr->what == mouseDown)
      {
        mouseXY = eventStrucPtr->where;
        GlobalToLocal(&mouseXY);
        if(FindControl(mouseXY,GetDialogWindow(dialogRef),&controlRef))
        {
          if(controlRef == gSlider1Ref || controlRef == gSlider2Ref)
          {
            TrackControl(controlRef,mouseXY,NULL);
            doDrawSliderValues(dialogRef,controlRef);
            handledEvent = true;
          }
          else if(controlRef == gSlider3Ref || controlRef == gSlider4Ref)
          {
            TrackControl(controlRef,mouseXY,gSliderActionFunction1UPP);
            handledEvent = true;
          }
          else if(controlRef == gSlider5Ref || controlRef == gSlider6Ref)
          {
            TrackControl(controlRef,mouseXY,gSliderActionFunction2UPP);
            handledEvent = true;
          }
        }
      }
    }
    else
    {
      handledEvent = StdFilterProc(dialogRef,eventStrucPtr,itemHit);
    }

    SetPort(oldPort);
  }

  return handledEvent;
}

// ******************************************************************** arrowsActionFunction

void  arrowsActionFunction(ControlRef controlRef,SInt16 partCode)
{
  Str255 theString;
  SInt32 controlValue;

  if(partCode)
  {
    controlValue = GetControlValue(controlRef);

    switch(partCode)
    {
      case kControlUpButtonPart:
        controlValue += 32;
        if(controlValue > GetControlMaximum(controlRef))
        {
          controlValue = GetControlMaximum(controlRef);
          return;
        }
      break;

      case kControlDownButtonPart:
        controlValue -= 32;
        if(controlValue < GetControlMinimum(controlRef))
        {
          controlValue = GetControlMinimum(controlRef);
          return;
        }
        break;
    }

    SetControlValue(controlRef,controlValue);

    NumToString((SInt32) controlValue,theString);
    doConcatPStrings(theString,"\pK");
    SetControlData(gCacheSizeControlRef,kControlEntireControl,kControlStaticTextTextTag,
                   theString[0],&theString[1]);
    Draw1Control(gCacheSizeControlRef);
  }
}

// ******************************************************************* sliderActionFunction1

void  sliderActionFunction1(ControlRef controlRef,SInt16 partCode)
{
  SInt16    staticTextItem;
  Str255    theString;
  DialogRef dialogRef;

  NumToString((SInt32) GetControlValue(controlRef),theString);

  dialogRef = GetDialogFromWindow(GetControlOwner(controlRef));

  if(controlRef == gSlider3Ref)
    staticTextItem = iSlider3StaticText;
  else if(controlRef == gSlider4Ref)
    staticTextItem = iSlider4StaticText;

  GetDialogItemAsControl(dialogRef,staticTextItem,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,theString[0],
                 &theString[1]);
  Draw1Control(controlRef);
}

// ******************************************************************* sliderActionFunction2

void  sliderActionFunction2(ControlRef controlRef,SInt16 partCode)
{
  SInt16 controlValue;
  Rect   theRect;

  controlValue = GetControlValue(controlRef);

  if(controlRef == gSlider5Ref)
  {
    gRedColour.red = 2 * controlValue;
    RGBForeColor(&gRedColour);
    SetRect(&theRect,220,177,236,193);
  }
  else if(controlRef == gSlider6Ref)
  {
    gBlueColour.blue = 2 * controlValue;
    RGBForeColor(&gBlueColour);
    SetRect(&theRect,220,198,236,214);
  }

  PaintRect(&theRect);
  RGBForeColor(&gBlackColour);
}

// *******************************************************************************************
// SmallControls.c
//*******************************************************************************************

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

#include "Controls3.h"

// ************************************************************************** doSmallControls

void  doSmallControls(void)
{
  OSStatus            osError;
  Str255              pascalString;
  CFStringRef         stringRef;
  Rect                contentRect        = {   0,  0,388,175 };
  Rect                staticTextRect     = {   8, 20, 86,144 };
  Rect                vertScrollBarRect  = {  -1,164,378,175 };
  Rect                horizScrollBarRect = { 377, -1,388,165 };
  Rect                tabRect            = { 106, 20,144,144 };
  Rect                sliderRect         = { 164, 20,182,145 };
  Rect                radioButtonRect    = { 200, 23,216,144 };
  Rect                checkBoxRect       = { 220, 23,236,144 };
  Rect                editTextRect       = { 300, 26,313,139 };
  Rect                pushButtonRect     = { 338, 21,356,144 };
  WindowRef           windowRef;
  ControlFontStyleRec controlFontStyleStruc;
  ControlRef          controlRef;
  ControlSize         controlSize = kControlSizeSmall;
  ControlTabEntry     tabArray[2];

  if(!(osError = CreateNewWindow(kFloatingWindowClass,kWindowStandardFloatingAttributes +
                                 kWindowResizableAttribute,&contentRect,
                                 &windowRef)))
  {
    ChangeWindowAttributes(windowRef,0,kWindowCloseBoxAttribute);
    RepositionWindow(windowRef,NULL,kWindowAlertPositionOnMainScreen);
    SetThemeWindowBackground(windowRef,kThemeBrushDialogBackgroundActive,false);
    SetPortWindowPort(windowRef);
    UseThemeFont(kThemeSmallSystemFont,smSystemScript);
    ShowWindow(windowRef);
  }
  else
  {
    SysBeep(10);
    return;
  }

  GetIndString(pascalString,rSmallControlsString,1);
  stringRef = CFStringCreateWithPascalString(NULL,pascalString,kCFStringEncodingMacRoman);

  controlFontStyleStruc.flags = kControlUseFontMask | kControlUseJustMask;
  controlFontStyleStruc.font   = kControlFontSmallSystemFont;
  controlFontStyleStruc.just  = teJustCenter;

  CreateStaticTextControl(windowRef,&staticTextRect,stringRef,&controlFontStyleStruc,
                           &controlRef);

  // ........................................................................................ scroll bars

  if((osError = CreateScrollBarControl(windowRef,&vertScrollBarRect,0,0,200,200,false,NULL,
                                       &controlRef)) == noErr)
    SetControlData(controlRef,kControlEntireControl,kControlSizeTag,sizeof(controlSize),
                   &controlSize);

  if((osError = CreateScrollBarControl(windowRef,&horizScrollBarRect,0,0,300,300,false,NULL,
                                       &controlRef)) == noErr)
    SetControlData(controlRef,kControlEntireControl,kControlSizeTag,sizeof(controlSize),
                   &controlSize);

  // ................................................................................................ tab

  tabArray[0].icon = tabArray[1].icon = NULL;
  tabArray[0].name = CFSTR("Tab 1");
  tabArray[1].name = CFSTR("Tab 2");
  tabArray[0].enabled = tabArray[1].enabled = true;

  CreateTabsControl(windowRef,&tabRect,kControlTabSizeSmall,kControlTabDirectionSouth,2,
                    tabArray,&controlRef);

  // ............................................................................................. slider

  if((osError = CreateSliderControl(windowRef,&sliderRect,0,0,100,
                                    kControlSliderPointsDownOrRight,5,false,NULL,
                                    &controlRef)) == noErr)
    SetControlData(controlRef,kControlEntireControl,kControlSizeTag,sizeof(controlSize),
                   &controlSize);

  // ....................................................................................... radio button

  if((osError = CreateRadioButtonControl(windowRef,&radioButtonRect,
                                         CFSTR("Small Radio Button"),1,false,
                                         &controlRef)) == noErr)
  {
    SetControlData(controlRef,kControlEntireControl,kControlSizeTag,sizeof(controlSize),
                   &controlSize);
    SetControlFontStyle(controlRef,&controlFontStyleStruc);
  }

  // ........................................................................................... checkbox

  if((osError = CreateCheckBoxControl(windowRef,&checkBoxRect,CFSTR("Small Checkbox"),1,false,
                                      &controlRef)) == noErr)
  {
    SetControlData(controlRef,kControlEntireControl,kControlSizeTag,sizeof(controlSize),
                   &controlSize);
    SetControlFontStyle(controlRef,&controlFontStyleStruc);
  }

  // .................................................................................. pop-up menu button

  if(controlRef = GetNewControl(cSmallPopup,windowRef))
    SetControlValue(controlRef,3);

  // .......................................................................................... edit text

  if((osError = CreateEditTextControl(windowRef,&editTextRect,CFSTR("Small Edit Text"),false,
                                      false,&controlFontStyleStruc,&controlRef)) == noErr)
    SetKeyboardFocus(windowRef,controlRef,kControlFocusNextPart);

  // ........................................................................................ push button

  if((osError = CreatePushButtonControl(windowRef,&pushButtonRect,CFSTR("Small Push Button"),
                                        &controlRef)) == noErr)
    SetControlFontStyle(controlRef,&controlFontStyleStruc);
```

```
}

// ****************************************************************** doMouseDownSmallControls

void  doMouseDownSmallControls(WindowRef windowRef,EventRecord *eventStrucPtr)
{
  SInt16      controlPartCode;
  ControlRef  controlRef = NULL;
  ControlKind controlKind;

  SetPortWindowPort(windowRef);
  GlobalToLocal(&eventStrucPtr->where);
  controlPartCode = FindControl(eventStrucPtr->where,windowRef,&controlRef);
  if(controlPartCode)
    TrackControl(controlRef,eventStrucPtr->where,NULL);

  if(controlRef)
  {
    GetControlKind(controlRef,&controlKind);
    if((controlKind.kind == kControlKindRadioButton) ||
        controlKind.kind == kControlKindCheckBox)
      SetControlValue(controlRef,!GetControlValue(controlRef));
  }
}

// *******************************************************************************************
```

Demonstration Program Controls3 Comments

When this program is run, the user should:

- Choose items from the Demonstration menu to view and operate the various controls. (Note that the Small Controls item is only available when the program is run on Mac OS X.)
- On Mac OS 8/9, choose Show Balloons from the Help menu and note the information in the help balloons as the cursor is moved over the various controls.
- On Mac OS X, hover the cursor over the various controls and note the information in the help tags.
- Click in the Finder and then back in the window/dialog, 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:
  - 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 Clock Controls Tab:

    - Change the keyboard focus using the tab key and mouse clicks.
    - Change the clock setting by typing and by using the integral little arrows.
    - Click the Extract push button to extract and display the clock settings.
- In the Group Boxes, Arrows, and Progress Bar dialog:
  - 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 chasing 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 (Mac OS 8/9).
- In the Text Boxes dialog, observe the auto scrolling variant and scroll the non-auto-scrolling variant.
- On Mac OS X, observe the small controls in the Small Controls floating window.

Control3.h

defines

Constants are established for the resource IDs and dialog item numbers associated with the various demonstrations.

typedefs

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

Control3.c

Controls3.c is simply the basic "engine" which supports the demonstration. There is nothing in this file which has not featured in previous demonstration programs.

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 function doBevelImagePictIconContent is called.

In the inGoAway case, the Bevel Buttons, Image Wells, Picture Controls, and Icon Controls window is disposed of.

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.

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 reference to the window's root control. The controls are then activated or deactivated en masse.

BevelImagePictIcon.c

BevelImagePictIcon.c 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 references to the various controls will be assigned) and stores the handle to this block in the window object.

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.) NewHandle creates the block for the variable of type BevelDocStruc and SetWRefCon stores the handle to the block as a reference constant in the window object.

Before ShowWindow is called to make the window visible, the function doGetControls is called to create the controls.

doGetControls

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

At the first line, if the program is running on Mac OS 8/9, CreateRootControl is called to create a root control for the window. On Mac OS 8/9, the first control created must be always be the root control (which is implemented as a user pane). This call is not necessary on Mac OS X because, on Mac OS X, root controls are created automatically for windows which have at least one control.

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

At the next block, if the program is running on Mac OS X, SetControlData is called to make the first four bevel button's corners rounded. (Mac OS X bevel buttons have square corners by default.)

At the next block, SetControlData is called twice to cause the text in the specified bevel buttons to be placed above the graphic.

At the next block, SetControlData is called 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 references to all the control objects 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, GetBevelButtonMenuValue is called 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 function 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 function 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 function 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 function 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 bevel button graphic alignment using the helper function SetBevelButtonGraphicAlignment, which facilitates finer adjustment of graphic placement that using SetControlData with the kControlBevelButtonGraphicAlignTag control data tag constant. Each time around the outer for loop, the alignment constant is changed. In the inner for loop, SetBevelButtonGraphicAlignment is called 53 times, with the two offset parameters incremented and Draw1Control called at each call. The alignment constant used during each pass through the outer for loop is drawn in the window header.

doTextAlignment

doTextAlignment is called to demonstrate the effect of the bevel button text alignment constants using the helper function SetBevelButtonTextAlignment, which facilitates finer adjustment of text placement that using SetControlData with the kControlBevelButtonGraphicAlignTag control data tag constant. Each time around the outer for loop, the alignment constant is changed. In the inner for loop, SetBevelButtonTextAlignmentis called 41 times, with the offset parameter incremented and Draw1Control called at each call. The alignment constant used during each pass through the outer for loop 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 the bevel button text placement constants. Within a for loop which increments the text placement constant, SetBevelButtonTextPlacement is called 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 either black or gray depending on whether the window is currently in front or in the background.

TabEditClock.c

TabEditClock.c creates a movable modal dialog in which is demonstrated a tab control, edit text controls and clock controls. A 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.c 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 embeds 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 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 Control 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 (Mac OS 8/9) or make it pulsing blue (Mac OS X). SetDialogTracksCursor will cause the Dialog Manager to change the cursor shape to the I-Beam cursor whenever the cursor is over an edit text item specified in the 'DITL' as an edit text primitive.

The first call to GetDialogItemAsControl gets a reference 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 universal procedure pointers for event filter, key filter, and edit text validator callback 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 reference 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 Controls tab, the Clock Controls user pane is hidden, the Edit Text Controls user pane is shown, and the keyboard focus is set to the first edit text control. If the tab hit was the Clock Controls tab, the Edit Text Controls 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 Controls pane, the image well control is re-drawn to erase any previous text and a function 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 a function 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 universal procedure pointers 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 reference 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 and modifiers parameters passed to this function by the control are first examined to determine whether (1) the character code represents the numeric characters between 0 and 9 inclusive or (2) whether the Command key was down. If either of these conditions is true, the function returns kControlKeyFilterPassKey. This means that, if the character is between 0 and 9 inclusive, it will be accepted. It also means that, if the Command key was down, non-numeric characters will be accepted as well. This latter is necessary to ensure that the user can edit the text in the edit text control using the usual Edit menu Command-key equivalents.

If a return does not occur at that point, the character code is further examined to determine whether it represents one of the arrow keys, the backspace key or the delete key. 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 callback 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.c

GroupArrowsProgress.c creates a movable modal dialog in which is demonstrated a checkbox group box, a pop-up group box, little arrows, a disclosure triangle, chasing 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.c 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 a chasing arrows control and a progress bar control, is appended to the dialog when the disclosure triangle is clicked.

doGroupArrowsProgress

doGroupArrowsProgress 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 universal procedure pointers for the application-defined event filter (callback) function and an application-defined action function for the little arrows control. (The event filter function is in the source code file Callbacks.c.)

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, functions 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 function 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 a function is called to extract 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 universal procedure pointers 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 three lines flip 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.

doChasingAndProgress

doChasingAndProgress further handles a hit in the disclosure triangle.

The first three lines flip 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 chasing 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 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 no 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 chasing arrows and progress bar.

Finally, SetControlData is called with the kControlStaticTextTextTag tag to change the text in the static text control 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 control 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.c.

Sliders.c

Sliders.c creates a movable modal dialog 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 universal procedure pointers for the application-defined event filter (callback) 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.c.)

The next two blocks creates universal procedure pointers 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 assigns references to the first four sliders to global variables and call a function to draw the respective control values in static text items located immediately below the sliders.

The following four lines assign references 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 universal procedure pointers 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 controls 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 reference, 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 DrawThemePlacard draw two Appearance Manager placard 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 placard 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 placard should be displayed with a deactivated appearance, the interior is left as drawn by DrawThemePlacard.

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 placard primitive and its interior is drawn.

TextBox.c

doTextBox is called when the user chooses TextBoxes from the Demonstration menu. It opens a movable modal dialog containing standard and scrolling text box controls.

Callbacks.c

Callbacks.c contains the event filter (callback) function used by all movable modal dialogs and three action functions, one for the little arrows, one for the vertical live-feedback sliders, 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 except that it intercepts mouse-down events in the little arrows and sliders and informs ModalDialog that it has handled those events.

After the call to SetPortDialogPort, 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 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 function 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 reference 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 coordinates based on the placard 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 coordinates based on the placard at the right of the slider.

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

SmallControls.c

doSmallControls

doSmallControls is called when the user chooses the Small Controls item in the Demonstration menu (Mac OS X only). It opens a floating window in which those controls with small versions are displayed.

Note that:

- A call to SetControlData with the kControlSizeTag tag is required to make the scroll bars small.
- The value passed in the third parameter of the CreateTabsControl call makes the tab control small.
- A call to SetControlData with the kControlSizeTag tag is required to make the slider control small.
- A call to SetControlData with the kControlSizeTag tag is required to make the radio button and checkbox controls small. In addition, a call to SetControlFontStyle is required to make the title small.
- The only way to create a pop-up menu button with both the menu and the title small is to create the control from a 'CNTL' resource, specifying the "use window font" variant. (Note that the window font is set to the small system font.)
- The edit text control is made small by, firstly, making the control's rectangle 12 pixels high and, secondly, changing the font to the small system font by passing the address of a ControlFontStyleRec structure in the sixth parameter of the CreateEditTextControl call.
- The push button is made small by, firstly, making the control's rectangle 17 pixels high and, secondly, calling SetControlFontStyle is to make the title small.