21

TEXT, TEXTEDIT, DATES, TIMES, AND NUMBERS

Demonstration Programs: MonoTextEdit and DateTimeNumbers

Introduction

The subject of text on the Macintosh is quite a complex matter, involving as it does QuickDraw, TextEdit, the Font Manager, the Text Utilities, the Script Manager, the Text Services Manager, Apple Type Services for Unicode Imaging, the Resource Manager, keyboard resources, and international resources. Part of that complexity arises from the fact that the system software supports many different writing systems, including Roman, Chinese, Japanese, Hebrew, and Arabic.[1]

Text on the Macintosh was touched on briefly at Chapter 12, which included descriptions of QuickDraw functions used for drawing text and for setting the font, style, size, and transfer mode. Chapter 15 contained a brief treatment of considerations applying to the printing of text. Chapter 26 addresses the Multilingual Text Engine (MLTE) introduced with Mac OS 9.

This chapter addresses:

- TextEdit, which you can use to provide your application with basic text editing and formatting capabilities.

  Note that the emphasis in this chapter is on monostyled TextEdit. With the introduction, with Mac OS 9, of the Multilingual Text Engine (see Chapter 26), it became all but inconceivable that programmers would ever again use multistyled TextEdit to provide their applications with multi-styled text editing capabilities. Accordingly, in the following, multistyled TextEdit is addressed only to the extent necessary to support an understanding of the display of non-editable multi-styled text, as in the Help dialog component of the demonstration program MonoTextEdit associated with this chapter.

- The formatting and display of dates, times, and numbers.

Before addressing those particular subjects, however, a brief overview of various closely related matters is appropriate.

[1] Some of the information in this chapter is valid only in the case of the Roman writing system.

More on Text

Characters, Character Sets and Codes, Glyphs, Typefaces, Styles, Fonts and Font Families

Characters and Character Sets and Codes

A **character** is a symbol which represents the concept of, for example, a lowercase "b", the number "2" or the arithmetic operator "+". A collection of characters is called a **character set**. Individual characters within a character set are identified by a **character code**.

The **Apple Standard Roman character set** is the fundamental character set for the Macintosh computer. As shown at Fig 1, it uses all character codes from `0x00` to `0xFF`. The Standard Roman character set is actually an extended version of the **ASCII character set**, which uses character codes from `0x00` to `0x7F` only, and which is highlighted at Fig 1.

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
x0	nul	dle	sp	0	@	P	`	p	Ä	ê	†	∞	¿	–	‡	
x1	soh	DC1	!	1	A	Q	a	q	Å	ë	°	±	¡	—	·	Ò
x2	stx	DC2	"	2	B	R	b	r	Ç	í	¢	≤	¬	“	‚	Ú
x3	etx	DC3	#	3	C	S	c	s	É	ì	£	≤	√	”	„	Û
x4	eot	DC4	$	4	D	T	d	t	Ñ	î	§	¥	ƒ	‘	‰	Ù
x5	enq	nak	%	5	E	U	e	u	Ö	ï	•	µ	≈	’	Â	ı
x6	ack	syn	&	6	F	V	f	v	Ü	ñ	¶	∂	∆	÷	Ê	ˆ
x7	bel	etb	'	7	G	W	g	w	á	ó	ß	∑	«	◊	Á	˜
x8	bs	can	(	8	H	X	h	x	à	ò	®	∏	»	ÿ	Ë	¯
x9	ht	em	)	9	I	Y	i	y	â	ô	©	π	…	Ÿ	È	˘
xA	lf	sub	*	:	J	Z	j	z	ä	ö	™	∫		⁄	Í	˙
xB	vt	esc	+	;	K	[	k	{	ã	õ	´	ª	À	€	Î	˚
xC	ff	fs	,	<	L	\	l	|	å	ú	¨	º	Ã	‹	Ï	¸
xD	cr	gs	-	=	M	]	m	}	ç	ù	≠	Ω	Õ	›	Ì	˝
xE	so	rs	.	>	N	^	n	~	é	û	Æ	æ	Œ	ﬁ	Ó	˛
xF	si	us	/	?	O	_	o	del	è	ü	Ø	ø	œ	ﬂ	Ô	ˇ

CONTROL CODES	ROMAN CHARACTERS	SCRIPT-SPECIFIC CHARACTERS
LOW ASCII RANGE		HIGH ASCII RANGE

FIG 1 - THE STANDARD ROMAN CHARACTER SET

Glyphs

The visual representation of a character on a display device is called a **glyph**. In other words, a glyph is the shape by which a character is represented. A specific character can be represented by many different shapes (that is, glyphs).

Two types of glyphs are used by the Font Manager: **bitmapped glyphs** and glyphs from **outline fonts**. A bitmapped glyph is a bitmap designed at a fixed size for a particular display device. An "outline" is a mathematical description of the glyph in terms of lines and curves, and is used by the Font Manager to create bitmaps at any size for any display device.

Typefaces

If all glyphs for a character set share certain design characteristics, they form a **typeface**. Typefaces have their own names, such as Arial, Geneva, or Times.

Styles

A specific variation in a glyphs appearance is called a **style**. On the Macintosh, available styles include plain, bold, italic, underline, outline, shadow, condensed, and extended. QuickDraw can add styles to bitmaps, or fonts can be designed a specific style, such as, for example, Arial Italic.

Fonts and Font Families

A **font** is a full set of glyphs in a specific typeface and style. All fonts have a font name, such as "Arial" or "Geneva", which is ordinarily the same name as the typeface from which it was derived. Except for fonts not in the plain style, the font's name includes the style (or styles), for example "Palatino Bold Italic".

Fonts on the Macintosh are resources. The resource types are as follows:

- Bitmapped font resources are of type `'FONT'` (the original resource type for fonts) and `'NFNT'` (bitmapped font). `'FONT'` and `'NFNT'` resources provide a separate bitmap for each glyph in each style and size.
- Outline font resources are of type `'sfnt'`. `'sfnt'` resources comprise glyphs in a particular typeface and style.

If multiple fonts of the same typeface are present, the Font Manager groups those fonts into **font families** of resource type `'FOND'`. A **font family ID** is the resource ID for a font family.

As an aside, most (though not all) fonts assign glyphs to character codes `0x20` to `0x7F` which visually define the characters associated with those codes.[2] However, there are differences in the glyphs assigned to the high-ASCII range. Indeed, some fonts do not actually include glyphs for all, or part, of the high-ASCII range.

Font Measurements

Fonts are either **monospaced** or **proportional**. All glyphs in a monospaced font are the same width. The glyphs in a proportional font have different widths, "m" being wider than "i", for example.

Base Line, Ascent Line and Descent Line

Most glyphs in a font sit on an imaginary line called the **base line**. The **ascent line** approximately corresponds with the tops of the uppercase letters of the font. The **descent line** usually corresponds to the bottom of descenders (the tails of glyphs like "j"). See Fig 2.

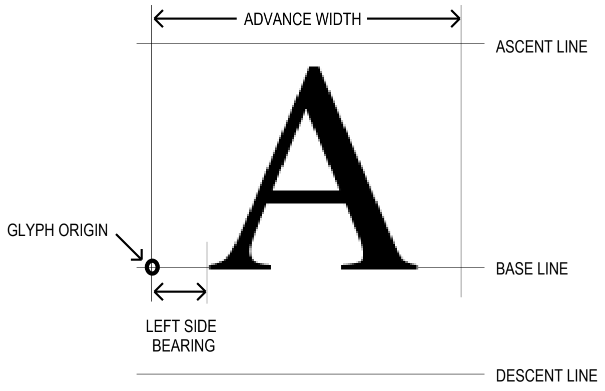


FIG 2 - FONT MEASUREMENT TERMS

Glyph Origin, Left Side Bearing, and Advance Width

The **glyph origin** is where QuickDraw begins drawing the glyph. The **left side bearing** is the white space between the glyph origin and the beginning of the glyph. The **advance width** is the full width of a glyph, measured from its origin to the origin of the next glyph. See Fig 2.

Font Size

Font size is the measurement, in **points**, from the base line of one line of text to the base line of the next line (assuming single-spaced text). A point is equivalent to 1/72 of an inch. The size of a font is often, but not always, the sum of the ascent, descent and **leading** (pronounced "ledding") values for a font. (The

[2] Fonts such as Zapf Dingbats assign glyphs of pictorial symbols to this range. as well as the low-ASCII range.

leading is the vertical space between the descent line of one line of single-spaced text and the ascent line of the next line.)

The Font Manager and QuickDraw

The Font Manager keeps track of all fonts available to an application and supports QuickDraw by providing the character bitmaps that QuickDraw needs. If QuickDraw requests a typeface that is not represented in the available fonts, the Font Manager substitutes one that is. Where necessary, QuickDraw scales the font to the requested size and applies the specified style.

Aspects of Text Editing — Caret Position, Text Offsets, Selection Range, Insertion Point, and Highlighting

Caret Position and Text Offset

In the world of text editing, the **caret** is defined as the blinking vertical bar that indicates the **insertion point** in text, and a **caret position** is a location on the screen that corresponds to an insertion point in memory. A caret position is always *between* glyphs on the screen. The caret is always positioned on the leading edge of the glyph corresponding to the character at the insertion point in memory. When a new character is inserted, the character at the insertion point, and all subsequent characters, are shifted forward one character position in memory.

The relationship between caret position, insertion point and offset is illustrated at Fig 3.

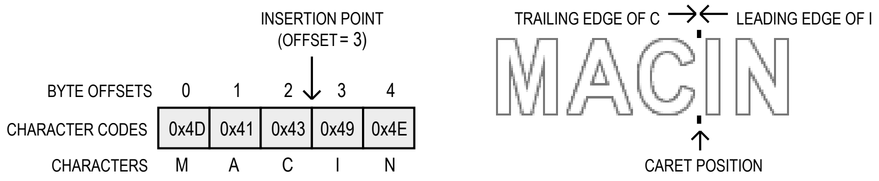


FIG 3 - CARET POSITION AND INSERTION POINT

Converting Screen Position to Text Offset

A mouse-down event can occur anywhere in a glyph; however, the caret position derived from that mouse-down must be an infinitely thin line between two glyphs.

As shown at Fig 4, a line of glyphs is divided into **mouse-down regions**, which, except at the end of the line, extend from the centre of one glyph to the centre of the next glyph. A click anywhere in a particular mouse-down region yields the same caret position.

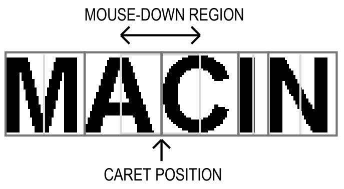


FIG 4 - INTERPRETING CARET POSITION FROM A MOUSE-DOWN EVENT

Selection Range and Insertion Points

The **selection range** is the sequence of zero or more contiguous characters where the next text editing operation is to occur. If a selection range contains zero characters, it is called an **insertion point**.

Highlighting

A selection range is typically marked by **highlighting**, that is, by drawing the glyphs with a coloured background. The limits of highlighting rectangles are measured in terms of caret position. For example, if the characters A, C, and I at Fig 3 were highlighted, the highlighting would extend from the leading edge of A (offset = 1) to the leading edge of N (offset = 4).

Outline Highlighting

Outline highlighting is the "framing" of text in the selection range in an inactive window. If there is no selection range, a gray, unblinking caret is displayed.

Keyboards and Text

Each keypress on a particular keyboard generates a value called a **raw key code**. The keyboard driver which handles the keypress uses the **key-map** (`'KMAP'`) **resource** to map the raw key code to a keyboard-independent **virtual key code**. It then uses the Event Manager and the **keyboard layout** (`'KCHR'`) **resource** to convert a virtual keycode into a character code. The character code is passed to your application in the event structure generated by the keypress.

Introduction to TextEdit

TextEdit is a collection of functions and data structures which you can use for the purposes of basic text formatting and editing. It was originally designed to handle edit text items in dialogs, and was subsequently enhanced to provide some of the more complex capabilities required of a basic text editor. That said, it should be understood that TextEdit was never intended to support all of the basic features generally required of a text editor (for example, tabs) and was never intended to manipulate lengthy text documents in excess of 32 KB. Indeed, the limit for documents created by TextEdit is 32,767 characters.

If you do not need to create large files and only need basic formatting capabilities, TextEdit provides a useful alternative to writing your own specialised text processing functions.

Editing Tasks Performed by TextEdit

The fundamental editing tasks which TextEdit can perform for your application are as follows:

- Selecting text by clicking and dragging the mouse, selecting words by double-clicking, and extending or shortening selections by Shift-clicking.
- Displaying the caret at the insertion point or highlighting the current text selection, as appropriate.
- Handling line breaking, that is, preventing a word from being split between lines.
- Cutting, copying, and pasting text within your application, and between your application and other applications.
- Managing the use of more than one font, text size, text colour, and text style from character to character.

TextEdit Options

You can use TextEdit at different levels of complexity.

Using TextEdit Indirectly

For the simplest level of text handling (that is, in dialogs), you need not even call TextEdit directly but rather use the Dialog Manager. The Dialog Manager, in turn, calls TextEdit to edit and display text.

Displaying Static Text

If you simply want to display one or more lines of static (non-editable) text, you can call `TETextBox`, which draws your text in the location you specify. `TETextBox` may be used to display text that you cannot edit. You do not need to create an **TextEdit structure** (see below) because `TETextBox` creates its own TextEdit structure. `TETextBox` draws the text in a rectangle whose size you specify in the coordinates of the current graphics port. Using the following constants, you can specify how text is aligned in the box:

Constant	***Description***
`teFlushDefault`	Default alignment according to primary line direction of the script system. (Left for Roman script system.)
`teCenter`	Centre alignment.
`teFlushRight`	Right alignment.
`teFlushLeft`	Left alignment.

Text Handling — Monostyled Text

If your application requires very basic text handling in a single typeface, style, and size, you probably only need **monostyled TextEdit**. You can use monostyled TextEdit with any single available font.

Text Handling — Multistyled Text

If your application requires a somewhat higher level of text handling (allowing the user to change typeface, style, and size within the document, for example), you can use **multistyled TextEdit**. However, as previously stated, mutistyled TextEdit has now been overshadowed by the introduction of the Multilingual Text Engine.

Caret Position and Movement in TextEdit

TextEdit always positions the caret where the next editing operation will occur. When TextEdit pastes text, it positions the caret after the newly pasted text. Assuming that the caret is not in the first or last line of text,TextEdit moves the caret up or down one line when the user presses the Up Arrow key or the Down Arrow key. (If the caret is on the first line, TextEdit moves the caret to the beginning of text on that line if the user presses the Up Arrow key,. If the caret is on the last line, TextEdit moves the caret to the end of the text on that line if the user presses the Down Arrow key.)[3]

Automatic Scrolling

One way for the user is to select large blocks of text is to click in the text and, holding the mouse button down, drag the cursor above, below, left of, or right of TextEdit's **view rectangle** (see below). While the mouse button remains down, and provided that your application has enabled automatic scrolling, TextEdit continually calls its **click loop function** to automatically scroll the text.

Although TextEdit's default click loop function automatically scrolls the text, it cannot adjust the scroll box/scroller position in an application's scroll bars to follow up the scrolling. The default click loop function can, however, be replaced with an application-defined click loop (callback) function which accommodates scroll bars.

TextEdit's Private, Null, and Style Scraps

Internally, TextEdit uses three scrap areas, namely, the **private scrap**, the **null scrap**, and the **style scrap**. The null scrap and the style scrap apply only to multistyled TextEdit.

The private scrap, which belongs to your application, is used for all cut, copy, and paste activity.

The null scrap is used by TextEdit to store **character attribute** information[4] associated with a null selection or text that is deleted by backspacing. (A null selection is an insertion point.)

When multistyled text is cut or copied, TextEdit copies character attribute information to the style scrap.

Text Alignment

Text **alignment** can be left-aligned, right-aligned, centred, or justified. Justified means aligned with both the left and right edges of TextEdit's **destination rectangle** (see below), and is achieved by spreading or compressing text to fit a given line width.

[3] TextEdit does not support the use of modifier keys, such as the Shift key, in conjunction with the arrow keys.

[4] The font, style, size, and colour aspects of text are collectively referred to as **character attributes**.

Primary TextEdit Data Structures

The primary data structures used by TextEdit are the TextEdit structure and the **dispatch structure**. Additional data structures are associated with multistyled TextEdit. This section describes the primary data structures only.

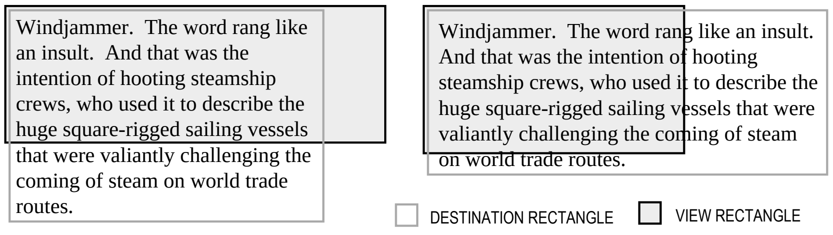
The TextEdit Structure

The TextEdit structure is the principal data structure used by TextEdit. This structure is the same regardless of whether the text is monostyled or multistyled, although some fields are used differently for multistyled TextEdit structures. The TextEdit structure is as follows:

```
struct TERec
{
  Rect           destRect;      // Destination rectangle.
  Rect           viewRect;      // View rectangle.
  Rect           selRect;       // Selection rectangle.
  short          lineHeight;    // Vert spacing of lines. -1 in multistyled.
  short          fontAscent;    // Font ascent. -1 in multistyled TextEdit structure.
  Point          selPoint;      // Point selected with the mouse.
  short          selStart;      // Start of selection range.
  short          selEnd;        // End of selection range.
  short          active;        // Set when structure is activated or deactivated.
  WordBreakUPP   wordBreak;     // Word break function.
  TEClickLoopUPP clickLoop;     // Click loop function.
  long           clickTime;     // (Used internally.)
  short          clickLoc;      // (Used internally.)
  long           caretTime;     // (Used internally.)
  short          caretState;    // (Used internally.)
  short          just;          // Text alignment.
  short          teLength;      // Length of text.
  Handle         hText;         // Handle to text to be edited.
  long           hDispatchRec;  // Handle to TextEdit dispatch structure.
  short          clikStuff;     // (Used internally)
  short          crOnly;        // If < 0, new line at Return only.
  short          txFont;     // Text font.   // If multistyled edit struct (txSize = -1),
  StyleField     txFace;     // Chara style. // these bytes are used as a handle
  SInt8          filler;     //              // to a style structure (TEStyleHandle).
  short          txMode;        // Pen mode.
  short          txSize;        // Font size. -1 in multistyled TextEdit structure.
  GrafPtr        inPort;        // Pointer to grafPort for this TextEdit structure.
  HighHookUPP    highHook;      // Used for text highlighting, caret appearance.
  CaretHookUPP   caretHook;     // Used from assembly language.
  short          nLines;        // Number of lines.
  short          lineStarts[16001];  // Positions of line starts.
};
typedef struct TERec TERec;
typedef TERec *TEPtr;
typedef TEPtr *TEHandle;
```

Field Descriptions

`destRect`	The destination rectangle (local coordinates), which is the area in which text is drawn (see Fig 5). The top of this rectangle determines the position of the first line of text and the two sides determine the beginning and the end of each line. The bottom of the rectangle varies as text is added or removed as a result of editing operations. The destination rectangle is central to the matter of scrolling text. When text is scrolled downwards, for example, you can think of the destination rectangle as being moved upwards through the view rectangle.
`viewRect`	The view rectangle (local coordinates), which is the area in which text is actually displayed (see Fig 5).



Windjammer. The word rang like an insult. And that was the intention of hooting steamship crews, who used it to describe the huge square-rigged sailing vessels that were valiantly challenging the coming of steam on world trade routes.
Windjammer. The word rang like an insult. And that was the intention of hooting steamship crews, who used it to describe the huge square-rigged sailing vessels that were valiantly challenging the coming of steam on world trade routes.
DESTINATION RECTANGLE
VIEW RECTANGLE

FIG 5 - DESTINATION AND VIEW RECTANGLES

`selRect`	The selection rectangle boundaries (local coordinates).
`lineHeight`	In a monostyled TextEdit structure, the vertical spacing of lines of text, that is, the distance from the ascent line of any one line of text to the ascent line of the next line of text. ***Multistyled TextEdit Structure.*** In a multistyled TextEdit structure, this field is set to -1, which indicates that line heights are calculated for each individual line of text.
`fontAscent`	In a monostyled TextEdit structure, the font ascent, that is, the vertical distance above the baseline the pen is positioned to begin drawing the caret or selection highlighting. (In the case of single-spaced text, the font ascent is the height of the text in pixels.) ***Multistyled TextEdit Structure.*** In a multistyled TextEdit structure, this field is set to -1, which indicates that font ascent is calculated for each individual line of text.
`selPoint`	The point selected with the mouse (local coordinates).
`selStart`	The byte offset of the start of the selection range. TextEdit initialises this field to 0 when you create an TextEdit structure.
`selEnd`	The byte offset of the end of the selection range. TextEdit initialises this field to 0 when you create an TextEdit structure. With both `selStart` and `selEnd` initialised to 0, the insertion point is placed at the beginning of the text.
`active`	Set when the TextEdit structure is activated and reset when the TextEdit structure is rendered inactive.
`wordBreak`	Universal procedure pointer to the word selection break function, which determines, firstly, the word that is highlighted when the user double-clicks in the text and, secondly, the position at which text is wrapped at the end of the line.
`clickLoop`	Universal procedure pointer to the click loop function, which is called repeatedly while the mouse button is held down within the text.
`just`	Text alignment (default, left, centre, or right).
`teLength`	The number of bytes in the text. The maximum allowable length is 32,767 bytes. When you create a TextEdit structure, TextEdit initialises this field to 0.
`hText`	A handle to the text. When you create a TextEdit structure, TextEdit initialises this field to point to a zero-length block in the application heap.
`hDispatchRec`	The handle to the TextEdit dispatch structure (see below). For internal use only.
`clikStuff`	TextEdit sets this field according to whether the most recent mouse-down event occurred on a glyph's leading or trailing edge. Used internally by TextEdit to determine a caret position.
`crOnly`	If the value in this field is positive, text wraps at the right edge of the destination rectangle. If the value is negative, text does *not* wrap.
`txFont`	In a monostyled TextEdit structure, the font of all the text in the TextEdit structure. (If you change the value, you should also change the `lineHeight` and `fontAscent` fields as appropriate.)

	Multistyled TextEdit Structure. In a multistyled TextEdit structure, if the `txSize` field (see below) is set to -1, this field combines with `txFace` and `filler` to hold a handle to the associated style structure.
`txFace`	In a monostyled TextEdit structure, the character attributes of all the text in a TextEdit structure. (If you change this value, you should also change the `lineHeight` and `fontAscent` fields as appropriate.)
	Multistyled TextEdit Structure. In a multistyled TextEdit structure, if the `txSize` field (see below) is set to -1, this field combines with `txFont` and `filler` to hold a handle to the associated style structure.
`txMode`	The pen mode of all the text.
`txSize`	In a monostyled TextEdit structure, this field is set to the size of the text in points.
	Multistyled TextEdit Structure. In a multistyled TextEdit structure, this field is set to is -1, indicating that the TextEdit structure contains associated character attribute information. The `txFont`, `txFace`, and `filler` fields combine to form a handle to the style structure in which this character attribute information is stored.
`inPort`	A pointer to the graphics port associated with the TextEdit structure.
`highHook`	A universal procedure pointer to the function that deals with text highlighting.
`caretHook`	Universal procedure pointer to the function that controls the appearance of the caret.
`numLines`	The number of lines of text.
`lineStarts`	A dynamic array which contains the character position of the first character in each line of the text. This array grows and shrinks, containing only as many elements as needed.

The Dispatch Structure

The `hDispatchRec` field of the TextEdit structure stores a handle to the dispatch structure. The dispatch structure is an internal data structure whose fields contain the addresses of functions which determine the way TextEdit behaves. You can modify TextEdit's default behaviour by replacing the address of a default function in the dispatch structure with the address of your own customized function.

Monostyled TextEdit

This section describes the use of TextEdit with monostyled text, that is, text with a single typeface, style, and size. Everything in this section also applies to using TextEdit with multistyled text except where otherwise indicated.

Creating, and Disposing of, a Monostyled TextEdit Structure

Creating a Monostyled TextEdit Structure

To use TextEdit functions, you must first create a TextEdit structure using `TENew`. `TENew` returns a handle to the newly-created monostyled TextEdit structure. You typically store the returned handle in a field of a document structure, the handle to which is typically stored in the application window's `refCon` field.

The required destination and view rectangles are specified in the `TENew` call. You should inset the destination rectangle at least four pixels from the left and right edges of the graphics port, making an additional allowance for scroll bars as appropriate. This will ensure that the first and last glyphs in each line are fully visible. You typically make the view rectangle equal to the destination rectangle. (If you do not want the text to be visible, specify a view rectangle off the screen.)

When a TextEdit structure is created, TextEdit initialises the TextEdit structure's fields based on values in the current graphics port object and on the type of TextEdit structure you create.

Disposing of an TextEdit Structure

Memory allocated for a TextEdit structure may be released by calling `TEDispose`.

Setting the Text

A new TextEdit structure does not contain any text until the user either opens an existing document or enters text via the keyboard. The following is concerned with existing documents.

`TESetText` may be used to specify the text to be edited. Alternatively, you can set the `hText` field of the TextEdit structure directly.

Calling TESetText

When a user opens a document, your application can load that document's text and then call `TESetText`. `TESetText` creates a copy of the text and stores the copy in the existing handle of the TextEdit structure's `hText` field.

You must pass the length of the text in the call to `TESetText`. `TESetText` uses this to reset the `teLength` field of the TextEdit structure, and to set the `selStart` and `selEnd` fields to the last byte offset of the text. `TESetText` also calculates the line breaks.

`TESetText` does not cause the text to be displayed immediately. You must call `InvalWindowRect` to force the text to be displayed at the next update event for the active window.

Changing the hText Field

The alternative of setting the `hText` field directly, replacing the existing handle with the handle of the new text, saves memory if you have a lot of text. When you use this method, you must also assign the length of the text to the `teLength` field of the TextEdit structure and call `TECalText` to recalculate the `lineStarts` array and `numLines` values.

Responding to Events

Activate Events

When your application receives an activate event (Classic event model) or `kEventWindowActivated` or `kEventWindowDeactivated` event type (Carbon event model), it should call `TEActivate` or `TEDeactivate` as appropriate.

A TextEdit structure which has been activated by `TEActivate` has its selection highlighted or, if there is no selection, has its caret displayed and blinking at the insertion point. A TextEdit structure which has been deactivated by `TEDeactivate` has its selection range outlined (if outline highlighting is enabled[5]) or, if there is no selection, has a grey, unblinking caret displayed at the insertion point.

Note that, when you use `TEClick` and `TESetSelect` (see below) to set the selection range or insertion point, the selection range is not highlighted, or the blinking caret is not displayed, until the TextEdit structure is activated. (However, if outline highlighting is enabled, the text of the selection range will be framed or a gray, unblinking caret will be displayed.)

Update Events — Calling TEUpdate

When your application receives an update event (Classic event model) or `kEventWindowDrawContent` or `kEventWindowUpdate` event type (Carbon event model), it should call `TEUpdate`. In addition, you should call `TEUpdate` after changing any fields of the TextEdit structure, or after any editing or scrolling operation, which alters the onscreen appearance of the text.

[5] Outline highlighting may be activated and deactivated using `TEFeatureFlag`.

Mouse-Down Events — Calling TEClick

On receipt of a mouse-down event that should be handled by TextEdit, your application must pass the event to `TEClick`. `TEClick` tells TextEdit that a mouse-down event has occurred. Before calling `TEClick`, however, your application must:

- Convert the mouse location from global coordinates to the local coordinates required by `TEClick`.
- Determine if the Shift key was down at the time of the event.

`TEClick` repeatedly calls the click loop function (see below) as long as the mouse button is held down and retains control until the button is released. The behaviour of `TEClick` depends on whether the Shift key was down at the time of the mouse-down event and on other user actions as follows:

User's Action	*Behaviour of TEClick*
Shift key down.	Extend the current selection range.
Shift key not down.	Remove highlighting from current selection range. Position the insertion point as close as possible to the location of the mouse click.
Mouse dragged.	Expand or shorten the selection range a character at a time. Keep control until the user releases the mouse button.
Double-click.	Extend the selection to include the entire word where the cursor is positioned.

Key-Down Events - Accepting Text Input

On receipt of a key-down event that should be handled by TextEdit, your application must call `TEKey` to accept the keyboard input. `TEKey` replaces the current selection range with the character passed to it and moves the insertion point just past the inserted character.

Depending on the requirements of your application, you may need to filter out certain character codes (for example, that for a Tab key press) so that they are not passed to `TEKey`. You should also check that the TextEdit limit of 32,767 bytes will not be exceeded by the insertion of the character before calling `TEKey` and you should call your scroll bar adjustment function immediately after the insertion.

Caret Blinking

To force the insertion point caret to blink, your application must call `TEIdle` at an interval equal to the value stored in the low-memory global `CaretTime`. You can retrieve this value by calling `GetCaretTime`. In Classic event model applications, you should set the `sleep` parameter in the `WaitNextEvent` call to this value and call `TEIdle` when `WaitNextEvent` returns `0` with a null event. In Carbon event model applications, you should install a timer set to fire at this interval and call `TEIdle` when the timer fires.

If there is more than one TextEdit structure associated with an active window, you must ensure that you pass `TEIdle` the handle to the currently active TextEdit structure. You should also check that the handle to be passed to `TEIdle` does not contain `NULL` before calling the function.

Cutting, Copying, Pasting, Inserting, and Deleting Text

Cutting, Copying, and Pasting

You can use TextEdit to cut, copy, and paste text within and between TextEdit structures, and across applications. The relevant functions, and their effect in the case of a monostyled TextEdit structure, are as follows:

Function	*Use To*	*Comments*
`TECut`	Cut text.	Copies the text to the TextEdit private scrap.
`TECopy`	Copy text.	Copies the text to the TextEdit private scrap.
`TEPaste`	Paste text.	Pastes from the TextEdit private scrap to the TextEdit structure.
`TEToScrap`	Copy TextEdit private scrap to the Carbon Scrap Manager's scrap.	Copying via the Carbon Scrap Manager's scrap is required if monostyled text is to be carried across applications.

`TEFromScrap`	Copy the Carbon Scrap Manager's scrap to TextEdit private scrap.	Copying via the Carbon Scrap Manager's scrap is required if text is to be carried across applications.
`TEGetScrapLength`	Determine the length of the text to be pasted.	Returns the size, in bytes, of the text in the private scrap.

You will need to call your vertical scroll bar adjustment function immediately after cut and paste operations. In addition, you will need to ensure that a paste will not cause the TextEdit limit of 32,767 bytes to be exceeded.

Inserting and Deleting Text

The following TextEdit functions are used to insert and delete monostyled text:

Function	*Use To*	*Comments*
`TEInsert`	Insert text into the TextEdit structure immediately before the selection range or insertion point.	Does not affect the selection range. Redraws the text if necessary.
`TEDelete`	Remove the selected range of text from the TextEdit structure.	Does not transfer the text to either TextEdit's private scrap or the Carbon Scrap Manager's scrap. Useful for implementing a **Clear** command. Redraws the remaining text if necessary.

You will need to call your vertical scroll bar adjustment function immediately after insertions and deletions. In addition, you will need to ensure that an insertion will not cause the TextEdit limit of 32,767 bytes to be exceeded.

Setting the Selection Range or Insertion Point

Using the `TESetSelect` function, your application can set the selection range or set the location of the insertion point. (For example, your application might use `TESetSelect` to locate the caret at the start of a data entry field where you want the user to enter a value.) `TESetSelect` changes the value in the `selStart` and `selEnd` fields of the TextEdit structure.

To set a selection range, you pass the byte offsets of the starting and ending characters in the `selStart` and `selEnd` parameters. To set the location of the insertion point, you pass the same values in the `selStart` and `selEnd` parameters. You can set the selection range (or insertion point) to any character position corresponding to byte offsets 0 to 32767.

To implement a **Select All** menu command, pass 0 in the `selStart` parameter and the value in the `teLength` field of the TextEdit structure in the `selEnd` parameter.

Enabling, Disabling, and Customising Automatic Scrolling

Enabling and Disabling

You can use the `TEAutoView` function to enable automatic scrolling (which, by default, is disabled). `TEAutoView` may also be used to disable automatic scrolling.

Customising

As previously stated, the default click loop (callback) function does not adjust the scroll bars as the text is scrolled, a situation that can be overcome by replacing the default click loop function with an application-defined click loop (callback) function which updates the scroll bars as it scrolls the text.

The `clickLoop` field of the TextEdit structure contains a universal procedure pointer to a click loop (callback) function, which is called continuously as long as the mouse button is held down. Installing your custom function involves a call to `TESetClickLoop` to assign the universal procedure pointer to the TextEdit structure's `clickLoop` field.

Scrolling Text

When a mouse-down event occurs in a scroll bar, your application must determine how far to scroll the text. The basic value for vertical scrolling of monostyled text is typically the value in the `lineHeight` field of the TextEdit structure, which can be used as the number of pixels to scroll for clicks in the Up and Down scroll arrows. For clicks in the gray areas/track, this value is typically multiplied by the number of text lines in the view rectangle minus 1. Scrolling by dragging the scroll box/scroller involves determining the number of text lines to scroll based on the current position of the top of the destination rectangle and the control value on mouse button release.

You pass the number of pixels to scroll in a call to `TEScroll` or `TEPinScroll`. (The difference between these two functions is that the latter stops scrolling when the last line is scrolled into the view rectangle.) The destination rectangle is offset by the amount you scroll.

Forcing the Selection Range Into the View

Your application can call `TESelView` to force the selection range to be displayed in the view rectangle. When automatic scrolling is enabled, `TESelView` scrolls the selection range into view, if necessary.

Setting Text Alignment

You can change the alignment of the entire text of a TextEdit structure by calling `TESetAlignment`. The following constants apply:

Constant	Description
`teFlushDefault`	Default alignment according to primary line direction of the script system. (Left for Roman script system.)
`teCenter`	Centre alignment.
`teFlushRight`	Right alignment.
`teFlushLeft`	Left alignment.

You should call the Window manager's `InvalWindowRect` function after you change the alignment so that the text is redrawn in the new alignment.

Saving and Opening TextEdit Documents

The demonstration program at Chapter 18 demonstrates opening and saving monostyled TextEdit documents.

Multistyled TextEdit

With the introduction, with Mac OS 9, of the Multilingual Text Editor (see Chapter 26), it became all but inconceivable that programmers would ever again use multistyled TextEdit to provide their applications with multi-styled text editing capabilities. That said, multistyled TextEdit may still be considered useful where the requirement is simply the display of non-editable multi-styled text, as in the Help dialog component of the demonstration program associated with this chapter.

This section addresses additional factors and considerations applying to multistyled TextEdit, but only to the extent necessary to support an understanding of those factors involved in the display of non-editable styled text, as in the Help dialog component of the demonstration program associated with this chapter.

Text With Multiple Styles — Style Runs, Text Segments, Font Runs, Character Attributes

Text that uses a variety of fonts, styles, sizes, and colours is referred to as **multistyled text**.

TextEdit organises multistyled text into **style runs**, which comprise a sets of contiguous characters which all share the same font, size, style, and colour characteristics. TextEdit tracks style runs in the data structures allocated for a multistyled TextEdit structure and uses this information to correctly display multistyled text.

The part of a style run that exists on a single line is called a **text segment**. A larger division than a style run is the **font run**, which comprises those characters which share the same font. The font, style, size, and colour aspects of text are collectively referred to as **character attributes**.

Additional TextEdit Data Structures for Multistyled Text

The TextEdit structure and the dispatch structure are the only data structures associated with monostyled text. However, when you allocate a multistyled TextEdit structure, a number of additional subsidiary data structures are created to support the text styling capabilities. The first of these additional data structures is the **style structure**, which stores the character attribute information for the text. (Recall that, when a multistyled TextEdit structure is created, the bytes at the `txFont`, `txFace`, and `filler` fields of the TextEdit structure contain a handle to the style structure.)

The additional data structures associated with a multistyled TextEdit structure are shown at Fig 6.

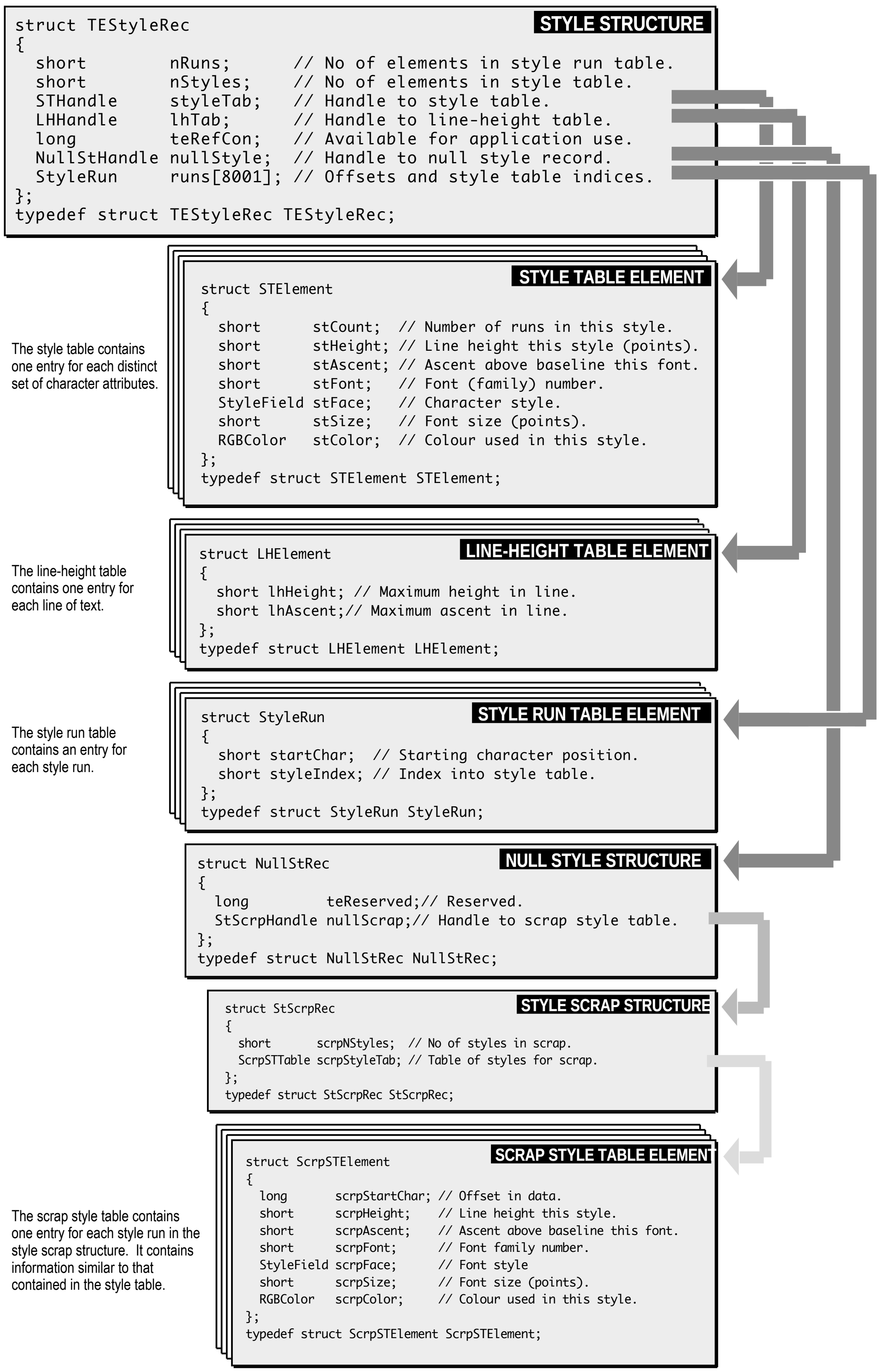
Creating a Multistyled TextEdit Structure

The multistyled TextEdit structure is created by calling `TEStyleNew`.

Inserting Text

The following describes `TEStyleInsert`, which is used to insert multistyled text:

Function	***Use To***	***Comments***
`TEStyleInsert`	Insert multistyled text into the TextEdit structure immediately before the selection range or insertion point.	Does not affect the selection range. Redraws the text if necessary. Applies the specified character attributes to the text. (You should create your own style scrap structure, specifying the style attributes to be inserted and applied to the text. These attributes are copied directly into the style structure's style table.)



```
struct TEStyleRec
{
  short        nRuns;      // No of elements in style run table.
  short        nStyles;    // No of elements in style table.
  STHandle     styleTab;   // Handle to style table.
  LHHandle     lhTab;      // Handle to line-height table.
  long         teRefCon;   // Available for application use.
  NullStHandle nullStyle;  // Handle to null style record.
  StyleRun     runs[8001]; // Offsets and style table indices.
};
typedef struct TEStyleRec TEStyleRec;
```

The style table contains one entry for each distinct set of character attributes.

```
struct STElement
{
  short      stCount;  // Number of runs in this style.
  short      stHeight; // Line height this style (points).
  short      stAscent; // Ascent above baseline this font.
  short      stFont;   // Font (family) number.
  StyleField stFace;   // Character style.
  short      stSize;   // Font size (points).
  RGBColor   stColor;  // Colour used in this style.
};
typedef struct STElement STElement;
```

The line-height table contains one entry for each line of text.

```
struct LHElement
{
  short lhHeight; // Maximum height in line.
  short lhAscent;// Maximum ascent in line.
};
typedef struct LHElement LHElement;
```

The style run table contains an entry for each style run.

```
struct StyleRun
{
  short startChar;  // Starting character position.
  short styleIndex; // Index into style table.
};
typedef struct StyleRun StyleRun;
```

```
struct NullStRec
{
  long         teReserved;// Reserved.
  StScrpHandle nullScrap;// Handle to scrap style table.
};
typedef struct NullStRec NullStRec;
```

```
struct StScrpRec
{
  short       scrpNStyles;  // No of styles in scrap.
  ScrpSTTable scrpStyleTab; // Table of styles for scrap.
};
typedef struct StScrpRec StScrpRec;
```

The scrap style table contains one entry for each style run in the style scrap structure. It contains information similar to that contained in the style table.

```
struct ScrpSTElement
{
  long       scrpStartChar; // Offset in data.
  short      scrpHeight;    // Line height this style.
  short      scrpAscent;    // Ascent above baseline this font.
  short      scrpFont;      // Font family number.
  StyleField scrpFace;      // Font style
  short      scrpSize;      // Font size (points).
  RGBColor   scrpColor;     // Colour used in this style.
};
typedef struct ScrpSTElement ScrpSTElement;
```

FIG 6 - THE STYLE STRUCTURE AND SUBSIDIARY DATA STRUCTURES

Formatting and Displaying Dates, Times, and Numbers

Preamble — The Text Utilities and International Resources

The Text Utilities

The **Text Utilities** are a collection of text-handling functions which you can use to, amongst other things, format numbers, currency, dates, and times.

International Resources

Many Text Utilities functions utilise the **international resources**, which define how different text elements are represented depending on the script system in use. The international resources relevant to formatting numbers, currency, dates, and times are as follows:

- ***Numeric Format Resource.*** The numeric format (`'itl0'`) resource contains short date and time formats, and formats for currency and numbers. It provides separators for decimals, thousands, and lists. It also contains the region code for this particular resource. Three of the several variations in short date and time formats are as follows:

System Software	Morning	Afternoon	Short Date
United States	`1:02 AM`	`1:02 PM`	`2/1/90`
Sweden	`01:02`	`13:02`	`90-01-01`
Germany	`1:02 Uhr`	`13:02 Uhr`	`2.1.1990`

- ***Long Date Format Resource.*** The long date format (`'itl1'`) resource specifies the long and abbreviated date formats for a particular region, including the names of days and months and the exact order of presentation of the elements. It also contains a region code for this particular resource. Three of the several variations of the long and abbreviated date formats are as follows:

System Software	Abbreviated Date	Long Date
United States	`Tue, Jan 2, 1990`	`Tuesday, January 2 1990`
French	`Mar 2 Jan 1990`	`Mardi 2 Janvier 1990`
Australian	`Tue, 2 Jan 1990`	`Tuesday, 2 January 1990`

- ***Tokens Resource.*** The tokens (`'itl4'`) resource contains, amongst other things, a table for formatting numbers. This table, which is called the **number parts table**, contains standard representations for the components of numbers and numeric strings. As will be seen, certain Text Utilities number formatting functions use the number parts table to create number strings in localised formats.

Date and Time

The Text Utilities functions which work with dates and times use information in the international resources to create different representations of date and time values. The Operating System provides functions that return the current date and time in numeric format. Text Utilities functions can then be used to convert these values into strings which can, in turn, be presented in the different international formats.

Date and Time Value Representations

The Operating System provides the following differing representations of date and time values:

Representation	Description
Standard date-time value.	A 32-bit integer representing the number of seconds between midnight, 1 January 1904 and the current time.
Long date-time value.	A 64-bit signed representation of data type `LongDateTime`. Allows for coverage of a longer time span than the standard date-time value, specifically, about 30,000 years.

Date-time structure.	Data type `DateTimeRec`. Includes integer fields for year, month, day, hour, minute, second, and day of week.
Long date-time structure.	Data type `LongDateRec`. Similar to the date-time structure, except that it adds several additional fields, including integer values for the era, day of the year, and week of the year. Allows for a longer time span than the date-time structure.

The date-time (`DateTimeRec`) and the long date-time (`LongDateRec`) structures are as follows:

```
struct DateTimeRec
{
  short year;
  short month;
  short day;
  short hour;
  short minute;
  short second;
  short dayOfWeek;
};

typedef struct DateTimeRec DateTimeRec;
```

```
union LongDateRec
{
  struct
  {
    short era;
    short year;
    short month;
    short day;
    short hour;
    short minute;
    short second;
    short dayOfWeek;
    short dayOfYear;
    short weekOfYear;
    short pm;
    short res1;
    short res2;
    short res3;
  } ld;
  short list[14];
  struct
  {
    short       eraAlt;
    DateTimeRec oldDate;
  } od;
};

typedef union LongDateRec LongDateRec;
```

Obtaining Date-Time Values and Structures

The Operating System Utilities provide the following two functions for obtaining date-time values and structures.

Function	Description
`GetDateTime`	Returns a standard date-time value.
`GetTime`	Returns a date-time structure.

Converting Between Values and Structures

The Operating System provides the following four functions for converting between the different date and time data types:

Function	Converts	To
`DateToSeconds`	Date-time structure.	Standard date-time value.
`SecondsToDate`	Standard date-time value.	Date-time structure.
`LongDateToSeconds`	Long date-time structure.	Long date-time value.
`LongSecondsToDate`	Long date-time value.	Long date-time structure.

Converting Date-Time Values Into Strings

The Text Utilities provide the following functions for converting from one of the numeric date-time representations to a formatted string.

Function	*Description*
`DateString`	Converts standard date-time value to a date string formatted according to the specified international resource.
`LongDateString`	Converts long date-time value to a date string formatted according to the specified international resource.
`TimeString`	Converts standard date-time value to a time string formatted according to the specified international resource.
`LongTimeString`	Converts long date-time values to a time string formatted according to the specified international resource.

Output Format — Date. When you use `DateString` and `LongDateString`, you can specify, in the `longFlag` parameter, an output format for the resulting date string. This format can be one of the following three values of the `DateForm` enumerated data type:

Value	*Date String Produced (Example)*	*Formatting Information Obtained From*
`shortDate`	`1/31/92`	Numeric format resource (`'itl0'`).
`abbrevDate`	`Fri, Jan 31, 1992`	Long date format resource (`'itl1'`).
`longDate`	`Friday, January 31, 1992`	Long date format resource (`'itl1'`).

Output Format — Time. When you use `TimeString` and `LongTimeString`, you can request an output format for the resulting time string by specifying either `true` or `false` in the `wantSeconds` parameter. `true` will cause seconds to be included in the string.

`DateString`, `LongDateString`, `TimeString` and `LongTimeString` use the date and time formatting information in the format resource that you specify in the resource handle (`intlHandle`) parameter. If you specify `NULL` for the value of the resource handle parameter, the appropriate format resource for the current script system is used.

Converting Date-Time Strings Into Internal Numeric Representation

The Text Utilities include functions which can parse date and time strings as entered by users and fill in the fields of a structure with the components of the date and time, including the month, day, year, hours, minutes, and seconds, extracted from the string.

Suppose your application needs to, say, convert a date and time string typed in by the user (for example, "`March 27, 1992, 08:14 p.m.`") into numeric representation. The following Text Utilities functions may be used to convert the string entered by the user into a long date-time structure:

Function	*Description*
`StringToDate`	Parses an input string for a date and creates an internal numeric representation of that date. Returns a status value indicating the confidence level for the success of the conversion. Expects a date specification, in one of the formats defined by the current script system, at the beginning of the string. Recognizes date strings in many formats, for example: "`September 1,1987`", "`1 Sept 87`", "`1/9/87`", and "`1 1987 Sept`".
`StringToTime`	Parses an input string for a time and creates an internal numeric representation of that time. Returns a status value indicating the confidence level for the success of the conversion. Expects a time specification, in a format defined by the current script system, at the beginning of the string.

You usually call `StringToDate` and `StringToTime` sequentially to parse the date and time values from an input string and fill in the fields of a long date-time structure. Note that `StringToDate` assigns to its `lengthUsed` parameter the number of bytes that it uses to parse the date. Use this value to compute the starting location of the text that you can pass to `StringToTime`.

The "confidence level" value returned by both `StringToDate` and `StringToTime` is of type `StringToDateStatus`, a set of bit values which have been OR'd together. The higher the resultant number, the lower the confidence level. Three of the twelve `StringToDateStatus` values, and their meanings, are as follows:

Value	*Meaning*
`fataldateTime`	Fatal error during the parse.
`dateTimeNotFound`	Valid date or time value not be found in string.
`sepNotIntlSep`	Valid date or time value found, but one or more of the separator characters in the string was not an expected separator character for the script system in use.

Date Cache Structure. Both `StringToDate` and `StringToTime` take a **date cache structure** as one of their parameters. A date cache structure (a data structure of type `DateCacheRec`) stores date conversion data used by the date and time conversion functions. You must declare a data cache structure in your application and initialise it by calling `InitDateCache` once, typically in your main program initialisation code.

Numbers

The Text Utilities provide several functions for converting between the internal numeric representation of a number and the output (or input) format of that number. You will need to perform these conversions when the user enters numbers for your application to use or when you present numbers to the user.

Integers

The simplest number conversion tasks involve integer values. The following Text Utilities functions may be used to convert an integer value to a numeric string and vice versa:

Function	*Description*
`NumToString`	Converts a long integer value into a string representation.
`StringToNum`	Converts a string representation of a number into a long integer value.

The range of values accommodated by these functions is -2,147,483,647 to 2,147,483,648. No comma insertion or other formatting is performed.

Number Format Specification Strings

Number format specification strings define the appearance of numeric strings. When you need to accommodate the differences in number output formats for different countries and regions, or when you are working with floating point numbers, you will need to use number format specification strings.

Parts. Each number format specification string contains up to three parts:

- The positive number format.
- The negative number format.
- The zero number format.

Each of these formats is applied to a numeric value of the corresponding type. When the specification string contains only one part, that part is used for all values. When in contains two parts, the first part is used for positive and zero values and the second part is used for negative values.

Elements. A number format specification string can contain the following elements:

- Number parts separators (, and .) for the decimal separator and the thousands separator.
- Literals to be included in the output. (Literals can be strings or brackets, braces and parentheses, and must be enclosed in quotation marks.)
- Digit place holders. (Digit place holders that you want displayed must be indicated by digit symbols. Zero digits (`0`) add leading zeroes whenever an input digit is not present. Skipping digits (`#`) only produce output characters when an input digit is present. Padding digits (`^`) are like zero digits except that a padding character such as a non-breaking space is used instead of leading zeros to pad the output string.)

- Quoting mechanisms for handling literals correctly.
- Symbol and sign characters.

Examples. The following shows several different number format specification strings and the output produced by each:

Number Format Specification String	***Numeric Value***	***Output Format***
`###,###.##;-###,###.##;0`	876543.21	`876,543.21`
`###,###.0##,###`	4321	`4,321.0`
`###,###.0##,###`	7.563489	`7.563,489`
`###;(000);^^^`	-1	`(001)`
`###.###`	5.234999	`5.235`
`###'CR';###'DB';''zero''`	1	`1CR`
`###'CR';###'DB';''zero''`	0	`'zero'`
`##%`	0.1	`10%`

Integer digits are always filled in from the right and decimal places are always filled in from the left. The following examples, in which a literal is included in the middle of the format strings, demonstrate this behaviour:

Number Format Specification String	***Numeric Value***	***Output Format***
`###'ab'###`	1	`1`
`###'ab'###`	123	`123`
`###'ab'###`	1234	`1ab1234`
`0.###'ab'###`	0.1	`0.1`
`0.###'ab'###`	0.123	`1.123`
`0.###'ab'###`	0.1234	`0.123ab4`

Overflow and Rounding. If the input string contains more digits than are specified in the number format specification string, an error (`formatOverflow`) will be generated. If the input string contains more decimal places than are specified in the number format specification string, the decimal portion is automatically rounded.

Converting Number Format Specification Strings to Internal Numeric Representations. With the required number format specification string defined, you must then convert the string into an internal numeric representation. The internal representation of format strings is stored in a `NumFormatString` structure. You use the following functions to convert a number format specification string to a `NumFormatString` structure and vice versa.

Function	***Description***
`StringToFormatRec`	Converts a number format specification string into a `NumFormatString` structure.
`FormatRecToString`	Convert a `NumFormatString` structure back to a number format specification string.

Number Parts Table. The internal numeric representation allows you to map the number into different output formats. One of the parameters taken by `StringToFormatRec` is a number parts table. The number parts table specifies which characters are used for certain purposes, such as separating parts of a number[6], in the format specification string.[7] As previously stated, the number parts table is contained in the `'itl4'` resource. A handle to the `'itl4'` resource may be obtained by a call to `GetIntlResourceTable`, specifying `iuNumberPartsTable` in the `tableCode` parameter.

6 For example, a thousands separator is a comma in Australia and a decimal point in France.

7 The `FormatRecToString` function also contains a number parts table parameter. By using a different table than was used in the call to `StringToFormatRec`, you can produce a number format specification string that specifies how numbers are formatted for a different region of the world. You use `FormatRecToString` when you want to display the number format specification string to the user for perusal or modification.

Converting Between Floating Point Numbers and Numeric Strings

Armed with a `NumFormatString` structure, you can convert floating point numbers into numeric strings and numeric strings into floating point numbers using the following functions:

Function	***Description***
`StringToExtended`	Using a `NumFormatString` structure and a number parts table, converts a numeric string to an 80-bit floating point value.
`ExtendedToString`	Using a `NumFormatString` structure and a number parts table, converts an 80-bit floating point number to a numeric string.

PowerPC Considerations. The PowerPC-based Macintosh follows the IEEE 754 standard for floating point arithmetic. In this standard, `float` is 32 bits and `double` is 64 bits. (Apple has added the 128 bit `long double` type.) However, the PowerPC floating point unit does not support Motorola's 80/96-bit `extended` type, and neither do the PowerPC numerics. To accommodate this, you can use Apple-supplied conversion utilities to move to and from `extended`. For example, the functions `x80tod` and `dtox80` (see the header file fp.h) can be used to directly transform 680x0 80-bit `extended` data types to `double` and back.

`StringToFormatRec`, `FormatRecToString`, `StringToExtended`, and `ExtendedToString` return a result of type `FormatStatus`, which is an integer value. The low byte is of type `FormatResultType`. Typical examples of the returned format status are as follows:

Value	***Meaning***
`fFormatOK`	The format of the input value is appropriate and the conversion was successful.
`fBestGuess`	The format of the input value is questionable. The result of the conversion may or may not be correct.
`fBadPartsTable`	The parts table is not valid.

Main TextEdit Constants, Data Types and Functions

Constants

Alignment

```
teFlushDefault          = 0
teCenter                = 1
teFlushRight            = -1
teFlushLeft             = -2
```

Feature or Bit Definitions for TEFeatureFlag feature Parameter

```
teFAutoScroll           = 0
teFTextBuffering        = 1
teFOutlineHilite        = 2
teFInlineInput          = 3
teFUseWhiteBackground   = 4
teFUseInlineInput       = 5
teFInlineInputAutoScroll = 6
```

Data Types

```
typedef char     Chars[32001];
typedef char     *CharsPtr;
typedef CharsPtr *CharsHandle;
```

TextEdit Structure

```
struct TERec
{
  Rect           destRect;      // Destination rectangle.
  Rect           viewRect;      // View rectangle.
  Rect           selRect;       // Selection rectangle.
  short          lineHeight;    // Vert spacing of lines. -1 in multistyled edit struct.
  short          fontAscent;    // Font ascent. -1 in multistyled TextEdit structure.
  Point          selPoint;      // Point selected with the mouse.
  short          selStart;      // Start of selection range.
  short          selEnd;        // End of selection range.
  short          active;        // Set when structure is activated or deactivated.
  WordBreakUPP   wordBreak;     // Word break function.
  TEClickLoopUPP clickLoop;     // Click loop function.
  long           clickTime;     // (Used internally.)
  short          clickLoc;      // (Used internally.)
  long           caretTime;     // (Used internally.)
  short          caretState;    // (Used internally.)
  short          just;          // Text alignment.
  short          teLength;      // Length of text.
  Handle         hText;         // Handle to text to be edited.
  long           hDispatchRec;  // Handle to TextEdit dispatch structure.
  short          clikStuff;     // (Used internally)
  short          crOnly;        // If < 0, new line at Return only.
  short          txFont;        // Text font.   // If multistyled edit struct (txSize = -1),
  StyleField     txFace;        // Chara style. // these bytes are used as a handle
  SInt8          filler;        //              // to a style structure (TEStyleHandle).
  short          txMode;        // Pen mode.
  short          txSize;        // Font size. -1 in multistyled TextEdit structure.
  GrafPtr        inPort;        // Pointer to grafPort for this TextEdit structure.
  HighHookUPP    highHook;      // Used for text highlighting, caret appearance.
  CaretHookUPP   caretHook;     // Used from assembly language.
  short          nLines;        // Number of lines.
  short          lineStarts[16001];  // Positions of line starts.
};
typedef struct TERec TERec;
typedef TERec *TEPtr;
typedef TEPtr *TEHandle;
```

Style Structure

```
struct TEStyleRec
{
  short        nRuns;          // Number of style runs.
  short        nStyles;        // Size of style table.
  STHandle     styleTab;       // Handle to style table.
  LHHandle     lhTab;          // Handle to line-height table.
  long         teRefCon;       // Reserved for application use.
  NullStHandle nullStyle;      // Handle to style set at null selection.
  StyleRun  runs[8001];        // ARRAY [0..8000] OF StyleRun.
};
typedef struct TEStyleRec TEStyleRec;
typedef TEStyleRec *TEStylePtr;
typedef TEStylePtr *TEStyleHandle;
```

Text Style Structure

```
struct TextStyle
{
  short        tsFont;         // Font (family) number.
  StyleField   tsFace;         // Character Style.
  short        tsSize;         // Size in point.
  RGBColor     tsColor;        // Absolute (RGB) color.
};
typedef struct TextStyle TextStyle;
typedef TextStyle *TextStylePtr;
typedef TextStylePtr *TextStyleHandle;
```

Functions

Creating and Disposing of TextEdit Structures

```
TEHandle    TENew(const Rect *destRect,const Rect *viewRect);
TEHandle    TEStyleNew(const Rect *destRect,const Rect *viewRect);
void        TEDispose(TEHandle hTE);
```

Activating and Deactivating a TextEdit Structure

```
void        TEActivate(TEHandle hTE);
void        TEDeactivate(TEHandle hTE);
```

Setting and Getting aTextEdit Structure's Text

```
void        TEKey(short key,TEHandle hTE);
void        TESetText(const void *text,long length,TEHandle hTE);
CharsHandle TEGetText(TEHandle hTE);
```

Setting the Caret and Selection Range

```
void        TEIdle(TEHandle hTE);
void        TEClick(Point pt,Boolean fExtend,TEHandle h);
void        TESetSelect(long selStart,long selEnd,TEHandle hTE);
```

Displaying and Scrolling Text

```
void        TESetAlignment(short just,TEHandle hTE);
void        TEUpdate(const Rect *rUpdate,TEHandle hTE);
void        TETextBox(const void *text,long length,const Rect *box,short just);
void        TECalText(TEHandle hTE);
long        TEGetHeight(long endLine,long startLine,TEHandle hTE);
void        TEScroll(short dh,short dv,TEHandle hTE);
void        TEPinScroll(short dh,short dv,TEHandle hTE);
void        TEAutoView(Boolean fAuto,TEHandle hTE);
void        TESelView(TEHandle hTE);
```

Modifying the Text of a TextEdit Structure

```
void        TEDelete(TEHandle hTE);
void        TEInsert(const void *text,long length,TEHandle hTE);
void        TEStyleInsert(const void *text,long length,StScrpHandle hST,TEHandle hTE);
void        TECut(TEHandle hTE);
void        TECopy(TEHandle hTE);
void        TEPaste(TEHandle hTE);
```

```
OSErr       TEFromScrap(void);
OSErr       TEToScrap(void);
```

Managing the TextEdit Private Scrap

```
Handle      TEScrapHandle(void);
long        TEGetScrapLength(void);
void        TESetScrapLength(long length);
```

Using Byte Offsets and Corresponding Points

```
short       TEGetOffset(Point pt,TEHandle hTE);
Point       TEGetPoint(short offset,TEHandle hTE);
```

Customising TextEdit

```
void        TESetClickLoop(TEClickLoopUPP clikProc,TEHandle hTE);;
void        TESetWordBreak(WordBreakUPP wBrkProc,TEHandle hTE);;
void        TECustomHook(TEIntHook which, UniversalProcPtr *addr,TEHandle hTE);
```

Additional TextEdit Features

```
short       TEFeatureFlag(short feature,short action,TEHandle hTE);
```

Main Constants, Data Types and Functions Relating to Dates, Times and Numbers

Constants

StringToDate and StringToTime Status Values

```
fatalDateTime      = 0x8000  Mask to a fatal error.
longDateFound      = 1       Mask to long date found.
leftOverChars      = 2       Mask to warn of left over characters.
sepNotIntlSep      = 4       Mask to warn of non-standard separators.
fieldOrderNotIntl  = 8       Mask to warn of non-standard field order.
extraneousStrings  = 16      Mask to warn of unparsable strings in text.
tooManySeps        = 32      Mask to warn of too many separators.
sepNotConsistent   = 64      Mask to warn of inconsistent separators.
tokenErr           = 0x8100  Mask for 'tokenizer err encountered'.
cantReadUtilities  = 0x8200
dateTimeNotFound   = 0x8400
dateTimeInvalid    = 0x8800
```

FormatResultType Values for Numeric Conversion Functions

```
fFormatOK          = 0
fBestGuess         = 1
fOutOfSynch        = 2
fSpuriousChars     = 3
fMissingDelimiter  = 4
fExtraDecimal      = 5
fMissingLiteral    = 6
fExtraExp          = 7
fFormatOverflow    = 8
fFormStrIsNAN      = 9
fBadPartsTable     = 10
fExtraPercent      = 11
fExtraSeparator    = 12
fEmptyFormatString = 13
```

Data Types

```
typedef short  StringToDateStatus;
typedef SInt8  DateForm;
typedef short  FormatStatus;
typedef Sint8  FormatResultType;
```

Data Cache Structure

```
struct DateCacheRecord
{
  short  hidden[256];  // Only for temporary use.
};
typedef struct DateCacheRecord DateCacheRecord;
typedef DateCacheRecord *DateCachePtr;
```

Number Format Specification Structure

```
struct NumFormatString
{
  UInt8 fLength;
  UInt8 fVersion;
  char  data[254];    // Private data.
};
typedef struct NumFormatString NumFormatString;
typedef NumFormatString NumFormatStringRec;
```

Functions

Getting Date-Time Values and Structures

```
void    GetDateTime(unsigned long *secs);
void    GetTime(DateTimeRec *d);
```

Converting Between Date-Time values and Structures

```
void    DateToSeconds(const DateTimeRec *d,unsigned long *secs);
void    SecondsToDate(unsigned long secs,DateTimeRec *d);
void    LongDateToSeconds(const LongDateRec *lDate,LongDateTime *lSecs);
void    LongSecondsToDate(LongDateTime *lSecs,LongDateRec *lDate);
```

Converting Date-Time Strings Into Internal Numeric Representation

```
OSErr   InitDateCache(DateCachePtr theCache);
StringToDateStatus  StringToDate(Ptr textPtr,long textLen,DateCachePtr theCache,
                    long *lengthUsed,LongDateRec *dateTime);
StringToDateStatus  StringToTime(Ptr textPtr,long textLen,DateCachePtr theCache,
                    long *lengthUsed,LongDateRec *dateTime);
```

Converting Long Date and Time Values Into Strings

```
void    DateString(long dateTime,DateForm longFlag,Str255 result);
void    TimeString(long dateTime,Boolean wantSeconds,Str255 result);
void    LongDateString(LongDateTime *dateTime,DateForm longFlag,Str255 result,
        Handle intlHandle);
void    LongTimeString(LongDateTime *dateTime,Boolean wantSeconds,Str255 result,
        Handle intlHandle);
```

Converting Between Integers and Strings

```
void    StringToNum(ConstStr255Param theString,long *theNum);
void    NumToString(long theNum,Str255 theString);
```

Using Number Format Specification Strings For International Number Formatting

```
FormatStatus  StringToFormatRec(ConstStr255Param inString,const NumberParts *partsTable,
              NumFormatString *outString)
FormatStatus  FormatRecToString(const NumFormatString *myCanonical,
              const NumberParts *partsTable,Str255 outString,TripleInt positions)
```

Converting Between Strings and Floating Point Numbers

```
FormatStatus  ExtendedToString(const extended80 x,const NumFormatString *myCanonical,
              const NumberParts *partsTable,Str255 outString)
FormatStatus  StringToExtended(ConstStr255Param source,const NumFormatString *myCanonical,
              const NumberParts *partsTable,extended80 *x)
```

Moving To and From Extended

```
void    x80told(const extended80 *x80,long double *x);
void    ldtox80(const long double *x,extended80 *x80);
double  x80tod(const extended80 *x80);
void    dtox80(const double *x, extended80 *x80);
```

Demonstration Program MonoTextEdit Listing

```
// *******************************************************************************************
// MonoTextEdit.c                                                          CARBON EVENT MODEL
// *******************************************************************************************
//
// This program demonstrates:
//
// •  A "bare-bones" monostyled text editor.
//
// •  A Help dialog which features the integrated scrolling of multistyled text and pictures.
//
// In the monostyled text editor demonstration, a panel is displayed at the bottom of all
// opened windows.  This panel displays the edit record length, number of lines, line height,
// destination rectangle (top), scroll bar/scroller value, and scroll bar/scroller maximum
// value.
//
// The bulk of the source code for the Help dialog is contained in the file HelpDialog.c.
// The dialog itself displays information intended to assist the user in adapting the Help
// dialog source code and resources to the requirements of his/her own application.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit, and Help dialog pop-up
//    menus (preload, non-purgeable).
//
// •  A 'CNTL' resources (purgeable) for the vertical scroll bar in the text editor window.
//
// •  'TEXT' and associated 'styl' resources (all purgeable) for the Help dialog.
//
// •  'PICT' resources (purgeable) for the Help dialog.
//
// •  A 'STR#' resource  (purgeable) containing error text strings.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// *******************************************************************************************

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

#include <Carbon.h>

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

#define rMenubar          128
#define mAppleApplication 128
#define  iAbout           1
#define  iHelp            2
#define mFile             129
#define  iNew             1
#define  iOpen            2
#define  iClose           4
#define  iSaveAs          6
#define  iQuit            12
#define mEdit             130
#define  iUndo            1
#define  iCut             3
#define  iCopy            4
#define  iPaste           5
#define  iClear           6
#define  iSelectAll       7
#define rVScrollbar       128
#define rErrorStrings     128
#define  eMenuBar         1
#define  eWindow          2
```

```
#define  eDocStructure    3
#define  eEditRecord      4
#define  eExceedChara     5
#define  eNoSpaceCut      6
#define  eNoSpacePaste    7
#define kMaxTELength      32767
#define kTab              0x09
#define kBackSpace        0x08
#define kForwardDelete    0x7F
#define kReturn           0x0D
#define kEscape           0x1B
#define topLeft(r)        (((Point *) &(r))[0])
#define botRight(r)       (((Point *) &(r))[1])

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

typedef struct
{
  TEHandle   textEditStrucHdl;
  ControlRef vScrollbarRef;
} docStructure, **docStructureHandle;

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

Boolean           gRunningOnX = false;
MenuID            gHelpMenu;
ControlActionUPP gScrollActionFunctionUPP;
TEClickLoopUPP    gCustomClickLoopUPP;
SInt16            gNumberOfWindows = 0;
SInt16            gOldControlValue;

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

void            main                 (void);
void            doPreliminaries      (void);
OSStatus        appEventHandler      (EventHandlerCallRef,EventRef,void *);
OSStatus        windowEventHandler   (EventHandlerCallRef,EventRef,void *);
void            doIdle               (void);
void            doKeyEvent           (SInt8);
void            scrollActionFunction (ControlRef,SInt16);
void            doInContent          (Point,Boolean);
void            doDrawContent        (WindowPtr);
void            doActivateDeactivate (WindowRef,Boolean);
WindowRef       doNewDocWindow       (void);
EventHandlerUPP doGetHandlerUPP      (void);
Boolean         customClickLoop      (void);
void            doSetScrollBarValue  (ControlRef,SInt16 *);
void            doAdjustMenus        (void);
void            doMenuChoice         (MenuID,MenuItemIndex);
void            doFileMenu           (MenuItemIndex);
void            doEditMenu           (MenuItemIndex);
SInt16          doGetSelectLength    (TEHandle);
void            doAdjustScrollbar    (WindowRef);
void            doAdjustCursor       (WindowRef);
void            doCloseWindow        (WindowRef);
void            doSaveAsFile         (TEHandle);
void            doOpenCommand        (void);
void            doOpenFile           (FSSpec);
void            doDrawDataPanel      (WindowRef);
void            doErrorAlert         (SInt16);
void            navEventFunction     (NavEventCallbackMessage,NavCBRecPtr,
                                      NavCallBackUserData);

extern void     doHelp               (void);

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

void  main(void)
{
```

```
MenuBarHandle menubarHdl;
SInt32        response;
MenuRef       menuRef;
EventTypeSpec applicationEvents[] = { { kEventClassApplication, kEventAppActivated    },
                                      { kEventClassCommand,     kEventProcessCommand  },
                                      { kEventClassMenu,        kEventMenuEnableItems },
                                      { kEventClassMouse,       kEventMouseMoved      } };

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

doPreliminaries();

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

menubarHdl = GetNewMBar(rMenubar);
if(menubarHdl == NULL)
  doErrorAlert(eMenuBar);
SetMenuBar(menubarHdl);
DrawMenuBar();

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

  menuRef = GetMenuRef(mAppleApplication);
  DeleteMenuItem(menuRef,iHelp);

  HMGetHelpMenu(&menuRef,NULL);
  InsertMenuItem(menuRef,"\pMonoTextEdit Help",0);
  gHelpMenu = GetMenuID(menuRef);

  gRunningOnX = true;
}
else
{
  menuRef = GetMenuRef(mFile);
  if(menuRef != NULL)
    SetMenuItemCommandID(menuRef,iQuit,kHICommandQuit);
}

// ............................................................. create universal procedure pointers

gScrollActionFunctionUPP = NewControlActionUPP((ControlActionProcPtr) scrollActionFunction);
gCustomClickLoopUPP      = NewTEClickLoopUPP((TEClickLoopProcPtr) customClickLoop);

// ............................................................... install application event handler

InstallApplicationEventHandler(NewEventHandlerUPP((EventHandlerProcPtr) appEventHandler),
                               GetEventTypeCount(applicationEvents),applicationEvents,
                               0,NULL);

// ................................................................................ install a timer

InstallEventLoopTimer(GetCurrentEventLoop(),0,TicksToEventTime(GetCaretTime()),
                      NewEventLoopTimerUPP((EventLoopTimerProcPtr) doIdle),NULL,
                      NULL);

// ........................................................................ open an untitled window

doNewDocWindow();

// ......................................................................... run application event loop
```

```
  RunApplicationEventLoop();
}

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

void  doPreliminaries(void)
{
  MoreMasterPointers(192);
  InitCursor();
}

// ************************************************************************** appEventHandler

OSStatus  appEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                          void * userData)
{
  OSStatus      result = eventNotHandledErr;
  UInt32        eventClass;
  UInt32        eventKind;
  HICommand     hiCommand;
  MenuID        menuID;
  MenuItemIndex menuItem;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  switch(eventClass)
  {
    case kEventClassApplication:
      if(eventKind == kEventAppActivated)
        SetThemeCursor(kThemeArrowCursor);
      break;

    case kEventClassCommand:
      if(eventKind == kEventProcessCommand)
      {
        GetEventParameter(eventRef,kEventParamDirectObject,typeHICommand,NULL,
                          sizeof(HICommand),NULL,&hiCommand);
        menuID = GetMenuID(hiCommand.menu.menuRef);
        menuItem = hiCommand.menu.menuItemIndex;
        if((hiCommand.commandID != kHICommandQuit) &&
           ((menuID >= mAppleApplication && menuID <= mEdit)) || menuID == gHelpMenu)
        {
          doMenuChoice(menuID,menuItem);
          result = noErr;
        }
      }
      break;

    case kEventClassMenu:
      if(eventKind == kEventMenuEnableItems)
      {
        GetWindowClass(FrontWindow(),&windowClass);
        if(windowClass == kDocumentWindowClass)
          doAdjustMenus();
        result = noErr;
      }
      break;

    case kEventClassMouse:
      if(eventKind == kEventMouseMoved)
      {
        GetWindowClass(FrontWindow(),&windowClass);
        if(windowClass == kDocumentWindowClass)
          doAdjustCursor(FrontWindow());
       result = noErr;
      }
      break;
  return result;
}

// ********************************************************************** windowEventHandler

OSStatus  windowEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                             void* userData)
{
  OSStatus        result = eventNotHandledErr;
  UInt32          eventClass;
  UInt32          eventKind;
  WindowRef       windowRef;
  UInt32          modifiers;
  Point           mouseLocation;
  Boolean         shiftKeyDown = false;
  ControlRef      controlRef;
  ControlPartCode controlPartCode;
  SInt8           charCode;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  switch(eventClass)
  {
    case kEventClassWindow:                                             // event class window
      GetEventParameter(eventRef,kEventParamDirectObject,typeWindowRef,NULL,sizeof(windowRef),
                        NULL,&windowRef);
      switch(eventKind)
      {
        case kEventWindowDrawContent:
          doDrawContent(windowRef);
          result = noErr;
          break;

        case kEventWindowActivated:
          doActivateDeactivate(windowRef,true);
          result = noErr;
          break;

        case kEventWindowDeactivated:
          doActivateDeactivate(windowRef,false);
          result = noErr;
          break;

        case kEventWindowClickContentRgn:
          GetEventParameter(eventRef,kEventParamMouseLocation,typeQDPoint,NULL,
                            sizeof(mouseLocation),NULL,&mouseLocation);
          SetPortWindowPort(FrontWindow());
          GlobalToLocal(&mouseLocation);
          GetEventParameter(eventRef,kEventParamKeyModifiers,typeUInt32,NULL,
                            sizeof(modifiers),NULL,&modifiers);
          if(modifiers & shiftKey)
            shiftKeyDown = true;
          doInContent(mouseLocation,shiftKeyDown);
          result = noErr;
          break;

        case kEventWindowClose:
          doCloseWindow(windowRef);
          result = noErr;
          break;
      }
      break;

    case kEventClassMouse:                                               // event class mouse
      switch(eventKind)
      {
        case kEventMouseDown:
          GetEventParameter(eventRef,kEventParamMouseLocation,typeQDPoint,NULL,
                            sizeof(mouseLocation),NULL,&mouseLocation);
          SetPortWindowPort(FrontWindow());
          GlobalToLocal(&mouseLocation);
          controlRef = FindControlUnderMouse(mouseLocation,FrontWindow(),&controlPartCode);
          if(controlRef)
          {
            gOldControlValue = GetControlValue(controlRef);
            TrackControl(controlRef,mouseLocation,gScrollActionFunctionUPP);
            result = noErr;
          }
        break;
      }
      break;

    case kEventClassKeyboard:                                         // event class keyboard
      switch(eventKind)
      {
        case kEventRawKeyDown:
        case kEventRawKeyRepeat:
          GetEventParameter(eventRef,kEventParamKeyMacCharCodes,typeChar,NULL,
                            sizeof(charCode),NULL,&charCode);
          GetEventParameter(eventRef,kEventParamKeyModifiers,typeUInt32,NULL,
                            sizeof(modifiers),NULL,&modifiers);
          if((modifiers & cmdKey) == 0)
            doKeyEvent(charCode);
          result = noErr;
          break;
      }
      break;
  }

  return result;
}

// *********************************************************************************** doIdle

void  doIdle(void)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;

  windowRef = FrontWindow();
  if(GetWindowKind(windowRef) == kApplicationWindowKind)
  {
    docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
    if(docStrucHdl != NULL)
      TEIdle((*docStrucHdl)->textEditStrucHdl);
  }
}

// ******************************************************************************** doKeyEvent

void  doKeyEvent(SInt8 charCode)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             selectionLength;

  if(charCode <= kEscape && charCode != kBackSpace && charCode != kReturn)
    return;

  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  if(charCode == kTab)
  {
    // Do tab key handling here if required.
  }
  else if(charCode == kForwardDelete)
  {
    selectionLength = doGetSelectLength(textEditStrucHdl);
    if(selectionLength == 0)
      (*textEditStrucHdl)->selEnd += 1;
    TEDelete(textEditStrucHdl);
    doAdjustScrollbar(windowRef);
  }
  else
  {
    selectionLength = doGetSelectLength(textEditStrucHdl);
    if(((*textEditStrucHdl)->teLength - selectionLength + 1) < kMaxTELength)
    {
      TEKey(charCode,textEditStrucHdl);
      doAdjustScrollbar(windowRef);
    }
    else
      doErrorAlert(eExceedChara);
  }

  doDrawDataPanel(windowRef);
}

// ******************************************************************** scrollActionFunction

void  scrollActionFunction(ControlRef controlRef,SInt16 partCode)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             linesToScroll;
  SInt16             controlValue, controlMax;

  windowRef = GetControlOwner(controlRef);
  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));;
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  controlValue = GetControlValue(controlRef);
  controlMax = GetControlMaximum(controlRef);

  if(partCode)
  {
    if(partCode != kControlIndicatorPart)
    {
      switch(partCode)
      {
        case kControlUpButtonPart:
        case kControlDownButtonPart:
          linesToScroll = 1;
          break;

        case kControlPageUpPart:
        case kControlPageDownPart:
          linesToScroll = (((*textEditStrucHdl)->viewRect.bottom -
                          (*textEditStrucHdl)->viewRect.top) /
                          (*textEditStrucHdl)->lineHeight) - 1;
          break;
      }

      if((partCode == kControlDownButtonPart) || (partCode == kControlPageDownPart))
        linesToScroll = -linesToScroll;

      linesToScroll = controlValue - linesToScroll;
      if(linesToScroll < 0)
        linesToScroll = 0;
      else if(linesToScroll > controlMax)
        linesToScroll = controlMax;
      SetControlValue(controlRef,linesToScroll);

      linesToScroll = controlValue - linesToScroll;
    }
    else
    {
      linesToScroll = gOldControlValue - controlValue;
      gOldControlValue = controlValue;
    }

    if(linesToScroll != 0)
    {
      TEScroll(0,linesToScroll * (*textEditStrucHdl)->lineHeight,textEditStrucHdl);
      doDrawDataPanel(windowRef);
    }
  }
}

// ***************************************************************************** doInContent

void  doInContent(Point mouseLocation,Boolean shiftKeyDown)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;

  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  if(PtInRect(mouseLocation,&(*textEditStrucHdl)->viewRect))
    TEClick(mouseLocation,shiftKeyDown,textEditStrucHdl);
}

// **************************************************************************** doDrawContent

void  doDrawContent(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  GrafPtr            oldPort;
  RgnHandle          visibleRegionHdl = NewRgn();
  Rect               portRect;

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

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

  UpdateControls(windowRef,visibleRegionHdl);

  GetWindowPortBounds(windowRef,&portRect);
  TEUpdate(&portRect,textEditStrucHdl);

  doDrawDataPanel(windowRef);

  DisposeRgn(visibleRegionHdl);
  SetPort(oldPort);
}

// ************************************************************************** doActivateDocWindow

void  doActivateDeactivate(WindowRef windowRef,Boolean becomingActive)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  if(becomingActive)
  {
    SetPortWindowPort(windowRef);

    (*textEditStrucHdl)->viewRect.bottom = ((((*textEditStrucHdl)->viewRect.bottom -
                                          (*textEditStrucHdl)->viewRect.top) /
                                          (*textEditStrucHdl)->lineHeight) *
                                          (*textEditStrucHdl)->lineHeight) +
                                          (*textEditStrucHdl)->viewRect.top;
    (*textEditStrucHdl)->destRect.bottom = (*textEditStrucHdl)->viewRect.bottom;

    TEActivate(textEditStrucHdl);
    ActivateControl((*docStrucHdl)->vScrollbarRef);
    doAdjustScrollbar(windowRef);
    doAdjustCursor(windowRef);
  }
  else
  {
    TEDeactivate(textEditStrucHdl);
    DeactivateControl((*docStrucHdl)->vScrollbarRef);
  }
}

// *************************************************************************** doNewDocWindow

WindowRef  doNewDocWindow(void)
{
  WindowRef          windowRef;
  OSStatus           osError;
  Rect               contentRect = { 100,100,400,595 };
  WindowAttributes   attributes  = kWindowStandardHandlerAttribute |
                                   kWindowStandardDocumentAttributes;
  docStructureHandle docStrucHdl;
  Rect               portRect, destAndViewRect;
  EventTypeSpec    windowEvents[] = { { kEventClassWindow,   kEventWindowDrawContent     },
                                      { kEventClassWindow,   kEventWindowActivated       },
                                      { kEventClassWindow,   kEventWindowDeactivated     },
                                      { kEventClassWindow,   kEventWindowClickContentRgn },
                                      { kEventClassWindow,   kEventWindowClose           },
                                      { kEventClassMouse,    kEventMouseDown             },
                                      { kEventClassKeyboard, kEventRawKeyDown            },
                                      { kEventClassKeyboard, kEventRawKeyRepeat          } };

  osError = CreateNewWindow(kDocumentWindowClass,attributes,&contentRect,&windowRef);
  if(osError != noErr)
  {
    doErrorAlert(eWindow);
    return NULL;
  }

  ChangeWindowAttributes(windowRef,0,kWindowResizableAttribute);
  RepositionWindow(windowRef,NULL,kWindowCascadeOnMainScreen);
  SetWTitle(windowRef,"\puntitled");
  SetPortWindowPort(windowRef);
  TextSize(10);

  InstallWindowEventHandler(windowRef,doGetHandlerUPP(),GetEventTypeCount(windowEvents),
                            windowEvents,0,NULL);

  if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
  {
    doErrorAlert(eDocStructure);
    return NULL;
  }
  SetWRefCon(windowRef,(SInt32) docStrucHdl);

  gNumberOfWindows ++;

  (*docStrucHdl)->vScrollbarRef = GetNewControl(rVScrollbar,windowRef);

  GetWindowPortBounds(windowRef,&portRect);
  destAndViewRect = portRect;
  destAndViewRect.right -= 15;
  destAndViewRect.bottom -= 15;
  InsetRect(&destAndViewRect,2,2);

  MoveHHi((Handle) docStrucHdl);
  HLock((Handle) docStrucHdl);

  if(!((*docStrucHdl)->textEditStrucHdl = TENew(&destAndViewRect,&destAndViewRect)))
  {
    DisposeWindow(windowRef);
    gNumberOfWindows --;
    DisposeHandle((Handle) docStrucHdl);
    doErrorAlert(eEditRecord);
    return NULL;
  }

  HUnlock((Handle) docStrucHdl);

  TESetClickLoop(gCustomClickLoopUPP,(*docStrucHdl)->textEditStrucHdl);
  TEAutoView(true,(*docStrucHdl)->textEditStrucHdl);
  TEFeatureFlag(teFOutlineHilite,1,(*docStrucHdl)->textEditStrucHdl);

  ShowWindow(windowRef);

  return windowRef;
}

// ************************************************************************** doGetHandlerUPP

EventHandlerUPP  doGetHandlerUPP(void)
{
  static EventHandlerUPP windowEventHandlerUPP;

  if(windowEventHandlerUPP == NULL)
    windowEventHandlerUPP = NewEventHandlerUPP((EventHandlerProcPtr) windowEventHandler);

  return windowEventHandlerUPP;
}

// ************************************************************************** customClickLoop

Boolean  customClickLoop(void)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  GrafPtr            oldPort;
  RgnHandle          oldClip;
  Rect               tempRect, portRect;
  Point              mouseXY;
  SInt16             linesToScroll = 0;

  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);
  oldClip = NewRgn();
  GetClip(oldClip);
  SetRect(&tempRect,-32767,-32767,32767,32767);
  ClipRect(&tempRect);

  GetMouse(&mouseXY);
  GetWindowPortBounds(windowRef,&portRect);

  if(mouseXY.v < portRect.top)
  {
    linesToScroll = 1;
    doSetScrollBarValue((*docStrucHdl)->vScrollbarRef,&linesToScroll);
    if(linesToScroll != 0)
      TEScroll(0,linesToScroll * ((*textEditStrucHdl)->lineHeight),textEditStrucHdl);
  }
  else if(mouseXY.v > portRect.bottom)
  {
    linesToScroll = -1;
    doSetScrollBarValue((*docStrucHdl)->vScrollbarRef,&linesToScroll);
    if(linesToScroll != 0)
      TEScroll(0,linesToScroll * ((*textEditStrucHdl)->lineHeight),textEditStrucHdl);
  }

  if(linesToScroll != 0)
    doDrawDataPanel(windowRef);

  SetClip(oldClip);
  DisposeRgn(oldClip);
  SetPort(oldPort);

  return true;
}

// ********************************************************************** doSetScrollBarValue

void  doSetScrollBarValue(ControlRef controlRef,SInt16 *linesToScroll)
{
  SInt16 controlValue, controlMax;

  controlValue = GetControlValue(controlRef);
  controlMax = GetControlMaximum(controlRef);

  *linesToScroll = controlValue - *linesToScroll;
  if(*linesToScroll < 0)
    *linesToScroll = 0;
  else if(*linesToScroll > controlMax)
    *linesToScroll = controlMax;

  SetControlValue(controlRef,*linesToScroll);
  *linesToScroll = controlValue - *linesToScroll;
}

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

void  doAdjustMenus(void)
{
  MenuRef            fileMenuHdl, editMenuHdl;
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  ScrapRef           scrapRef;
  OSStatus           osError;
  ScrapFlavorFlags   scrapFlavorFlags;

  fileMenuHdl = GetMenuRef(mFile);
  editMenuHdl = GetMenuRef(mEdit);

  if(gNumberOfWindows > 0)
  {
    windowRef = FrontWindow();
    docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
    textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
    EnableMenuItem(fileMenuHdl,iClose);

    if((*textEditStrucHdl)->selStart < (*textEditStrucHdl)->selEnd)
    {
      EnableMenuItem(editMenuHdl,iCut);
      EnableMenuItem(editMenuHdl,iCopy);
      EnableMenuItem(editMenuHdl,iClear);
    }
    else
    {
      DisableMenuItem(editMenuHdl,iCut);
      DisableMenuItem(editMenuHdl,iCopy);
      DisableMenuItem(editMenuHdl,iClear);
    }

    GetCurrentScrap(&scrapRef);

    osError = GetScrapFlavorFlags(scrapRef,kScrapFlavorTypeText,&scrapFlavorFlags);
    if(osError == noErr)
      EnableMenuItem(editMenuHdl,iPaste);
    else
      DisableMenuItem(editMenuHdl,iPaste);

    if((*textEditStrucHdl)->teLength > 0)
    {
      EnableMenuItem(fileMenuHdl,iSaveAs);
      EnableMenuItem(editMenuHdl,iSelectAll);
    }
    else
    {
      DisableMenuItem(fileMenuHdl,iSaveAs);
      DisableMenuItem(editMenuHdl,iSelectAll);
    }
  }
  else
  {
    DisableMenuItem(fileMenuHdl,iClose);
    DisableMenuItem(fileMenuHdl,iSaveAs);
    DisableMenuItem(editMenuHdl,iClear);
    DisableMenuItem(editMenuHdl,iSelectAll);
  }

  DrawMenuBar();
}

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

void  doMenuChoice(MenuID menuID, MenuItemIndex menuItem)
{
  if(menuID == 0)
    return;

  if(gRunningOnX)
    if(menuID == gHelpMenu)
      if(menuItem == 1)
        doHelp();

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

    case mFile:
      doFileMenu(menuItem);
      break;

    case mEdit:
      doEditMenu(menuItem);
      break;
  }
}

// ******************************************************************************* doFileMenu

void  doFileMenu(MenuItemIndex menuItem)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;

  switch(menuItem)
  {
    case iNew:
      if(windowRef = doNewDocWindow())
        ShowWindow(windowRef);
      break;

    case iOpen:
      doOpenCommand();
      doAdjustScrollbar(FrontWindow());
      break;

    case iClose:
      doCloseWindow(FrontWindow());
      break;

    case iSaveAs:
      docStrucHdl = (docStructureHandle) (GetWRefCon(FrontWindow()));
      textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
      doSaveAsFile(textEditStrucHdl);
      break;
  }
}

// ******************************************************************************* doEditMenu

void  doEditMenu(MenuItemIndex menuItem)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt32             totalSize, contigSize, newSize;
  SInt16             selectionLength;
  ScrapRef           scrapRef;
  Size               sizeOfTextData;

  windowRef = FrontWindow();

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  switch(menuItem)
  {
    case iUndo:
      break;

    case iCut:
      if(ClearCurrentScrap() == noErr)
      {
        PurgeSpace(&totalSize,&contigSize);
        selectionLength = doGetSelectLength(textEditStrucHdl);
        if(selectionLength > contigSize)
          doErrorAlert(eNoSpaceCut);
        else
        {
          TECut(textEditStrucHdl);
          doAdjustScrollbar(windowRef);
          TEToScrap();
          if(TEToScrap() != noErr)
            ClearCurrentScrap();
        }
      }
      break;

    case iCopy:
      if(ClearCurrentScrap() == noErr)
        TECopy(textEditStrucHdl);
      TEToScrap();
      if(TEToScrap() != noErr)
        ClearCurrentScrap();
      break;

    case iPaste:
      GetCurrentScrap(&scrapRef);;
      GetScrapFlavorSize(scrapRef,kScrapFlavorTypeText,&sizeOfTextData);
      newSize = (*textEditStrucHdl)->teLength + sizeOfTextData;
      if(newSize > kMaxTELength)
        doErrorAlert(eNoSpacePaste);
      else
      {
        if(TEFromScrap() == noErr)
        {
          TEPaste(textEditStrucHdl);
          doAdjustScrollbar(windowRef);
        }
      }
      break;

    case iClear:
      TEDelete(textEditStrucHdl);
      doAdjustScrollbar(windowRef);
      break;

    case iSelectAll:
      TESetSelect(0,(*textEditStrucHdl)->teLength,textEditStrucHdl);
      break;
  }

  doDrawDataPanel(windowRef);
}

// ************************************************************************* doGetSelectLength

SInt16  doGetSelectLength(TEHandle textEditStrucHdl)
{
  SInt16 selectionLength;

  selectionLength = (*textEditStrucHdl)->selEnd - (*textEditStrucHdl)->selStart;
  return selectionLength;
}

// ************************************************************************* doAdjustScrollbar

void  doAdjustScrollbar(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             numberOfLines, controlMax, controlValue;

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));;
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
  numberOfLines = (*textEditStrucHdl)->nLines;
  if(*(*(*textEditStrucHdl)->hText + (*textEditStrucHdl)->teLength - 1) == kReturn)
    numberOfLines += 1;

  controlMax = numberOfLines - (((*textEditStrucHdl)->viewRect.bottom -
               (*textEditStrucHdl)->viewRect.top) /
               (*textEditStrucHdl)->lineHeight);
  if(controlMax < 0)
    controlMax = 0;
  SetControlMaximum((*docStrucHdl)->vScrollbarRef,controlMax);

  controlValue = ((*textEditStrucHdl)->viewRect.top - (*textEditStrucHdl)->destRect.top) /
                  (*textEditStrucHdl)->lineHeight;
  if(controlValue < 0)
    controlValue = 0;
  else if(controlValue > controlMax)
    controlValue = controlMax;

  SetControlValue((*docStrucHdl)->vScrollbarRef,controlValue);

  SetControlViewSize((*docStrucHdl)->vScrollbarRef,(*textEditStrucHdl)->viewRect.bottom -
                     (*textEditStrucHdl)->viewRect.top);

  TEScroll(0,((*textEditStrucHdl)->viewRect.top - (*textEditStrucHdl)->destRect.top) -
               (GetControlValue((*docStrucHdl)->vScrollbarRef) *
              (*textEditStrucHdl)->lineHeight),textEditStrucHdl);
}

// *************************************************************************** doAdjustCursor

void  doAdjustCursor(WindowRef windowRef)
{
  GrafPtr   oldPort;
  RgnHandle arrowRegion, iBeamRegion;
  Rect      portRect, cursorRect;
  Point     mouseXY;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  arrowRegion = NewRgn();
  iBeamRegion = NewRgn();
  SetRectRgn(arrowRegion,-32768,-32768,32766,32766);

  GetWindowPortBounds(windowRef,&portRect);
  cursorRect = portRect;
  cursorRect.bottom -= 15;
  cursorRect.right  -= 15;
  LocalToGlobal(&topLeft(cursorRect));
  LocalToGlobal(&botRight(cursorRect));

  RectRgn(iBeamRegion,&cursorRect);
  DiffRgn(arrowRegion,iBeamRegion,arrowRegion);

  GetGlobalMouse(&mouseXY);

  if(PtInRgn(mouseXY,iBeamRegion))
    SetThemeCursor(kThemeIBeamCursor);
  else
    SetThemeCursor(kThemeArrowCursor);

  DisposeRgn(arrowRegion);
  DisposeRgn(iBeamRegion);

  SetPort(oldPort);
}

// **************************************************************************** doCloseWindow
```

```
void  doCloseWindow(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));;

  DisposeControl((*docStrucHdl)->vScrollbarRef);
  TEDispose((*docStrucHdl)->textEditStrucHdl);
  DisposeHandle((Handle) docStrucHdl);
  DisposeWindow(windowRef);

  gNumberOfWindows --;
}

// ***************************************************************************** doSaveAsFile

void  doSaveAsFile(TEHandle textEditStrucHdl)
{
  OSErr            osError = noErr;
  NavDialogOptions dialogOptions;
  NavEventUPP      navEventFunctionUPP;
  WindowRef        windowRef;
  OSType           fileType;
  NavReplyRecord   navReplyStruc;
  AEKeyword        theKeyword;
  DescType         actualType;
  FSSpec           fileSpec;
  SInt16           fileRefNum;
  Size             actualSize;
  SInt32           dataLength;
  Handle           editTextHdl;

  osError = NavGetDefaultDialogOptions(&dialogOptions);

  if(osError == noErr)
  {
    windowRef = FrontWindow();

    fileType = 'TEXT';

    navEventFunctionUPP = NewNavEventUPP((NavEventProcPtr) navEventFunction);
    osError = NavPutFile(NULL,&navReplyStruc,&dialogOptions,navEventFunctionUPP,fileType,
                         'kjBb',NULL);
    DisposeNavEventUPP(navEventFunctionUPP);

    if(navReplyStruc.validRecord && osError == noErr)
    {
      if((osError = AEGetNthPtr(&(navReplyStruc.selection),1,typeFSS,&theKeyword,
                                &actualType,&fileSpec,sizeof(fileSpec),&actualSize)) == noErr)

      {
        if(!navReplyStruc.replacing)
        {
          osError = FSpCreate(&fileSpec,'kjBb',fileType,navReplyStruc.keyScript);
          if(osError != noErr)
          {
            NavDisposeReply(&navReplyStruc);
          }
        }

        if(osError == noErr)
          osError = FSpOpenDF(&fileSpec,fsRdWrPerm,&fileRefNum);

        if(osError == noErr)
        {
          SetWTitle(windowRef,fileSpec.name);
          dataLength = (*textEditStrucHdl)->teLength;
          editTextHdl = (*textEditStrucHdl)->hText;
          FSWrite(fileRefNum,&dataLength,*editTextHdl);
        }

        NavCompleteSave(&navReplyStruc,kNavTranslateInPlace);
      }

      NavDisposeReply(&navReplyStruc);
    }
  }
}

// ****************************************************************************** doOpenCommand

void  doOpenCommand(void)
{
  OSErr            osError  = noErr;
  NavDialogOptions dialogOptions;
  NavEventUPP      navEventFunctionUPP;
  NavReplyRecord   navReplyStruc;
  SInt32           index, count;
  AEKeyword        theKeyword;
  DescType         actualType;
  FSSpec           fileSpec;
  Size             actualSize;
  FInfo            fileInfo;

  osError = NavGetDefaultDialogOptions(&dialogOptions);

  if(osError == noErr)
  {
    navEventFunctionUPP = NewNavEventUPP((NavEventProcPtr) navEventFunction);
    osError = NavGetFile(NULL,&navReplyStruc,&dialogOptions,navEventFunctionUPP,NULL,NULL,
                         NULL,NULL);
    DisposeNavEventUPP(navEventFunctionUPP);

    if(osError == noErr && navReplyStruc.validRecord)
    {
      osError = AECountItems(&(navReplyStruc.selection),&count);
      if(osError == noErr)
      {
        for(index=1;index<=count;index++)
        {
          osError = AEGetNthPtr(&(navReplyStruc.selection),index,typeFSS,&theKeyword,
                                &actualType,&fileSpec,sizeof(fileSpec),&actualSize);
          {
            if((osError = FSpGetFInfo(&fileSpec,&fileInfo)) == noErr)
              doOpenFile(fileSpec);
          }
        }
      }

      NavDisposeReply(&navReplyStruc);
    }
  }
}

// ********************************************************************************* doOpenFile

void  doOpenFile(FSSpec fileSpec)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             fileRefNum;
  SInt32             textLength;
  Handle             textBuffer;

  if((windowRef = doNewDocWindow()) == NULL)
    return;
  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));;
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  SetWTitle(windowRef,fileSpec.name);

  FSpOpenDF(&fileSpec,fsCurPerm,&fileRefNum);

  SetFPos(fileRefNum,fsFromStart,0);
  GetEOF(fileRefNum,&textLength);

  if(textLength > 32767)
    textLength = 32767;

  textBuffer = NewHandle((Size) textLength);

  FSRead(fileRefNum,&textLength,*textBuffer);

  MoveHHi(textBuffer);
  HLock(textBuffer);

  TESetText(*textBuffer,textLength,textEditStrucHdl);

  HUnlock(textBuffer);
  DisposeHandle(textBuffer);

  FSClose(fileRefNum);

  (*textEditStrucHdl)->selStart = 0;
  (*textEditStrucHdl)->selEnd = 0;

  doDrawContent(windowRef);
}

// ************************************************************************** doDrawDataPanel

void  doDrawDataPanel(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  RGBColor           whiteColour = { 0xFFFF, 0xFFFF, 0xFFFF };
  RGBColor           blackColour = { 0x0000, 0x0000, 0x0000 };
  RGBColor           blueColour = { 0x1818, 0x4B4B, 0x8181 };
  ControlRef         controlRef;
  Rect               panelRect;
  Str255             textString;

  SetPortWindowPort(windowRef);

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));;
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
  controlRef = (*docStrucHdl)->vScrollbarRef;

  MoveTo(0,282);
  LineTo(495,282);

  RGBForeColor(&whiteColour);
  RGBBackColor(&blueColour);
  SetRect(&panelRect,0,283,495,300);
  EraseRect(&panelRect);

  MoveTo(3,295);
  DrawString("\pteLength               nLines          lineHeight");

  MoveTo(225,295);
  DrawString("\pdestRect.top              controlValue           contrlMax");

  SetRect(&panelRect,47,284,88,299);
  EraseRect(&panelRect);
  SetRect(&panelRect,124,284,149,299);
  EraseRect(&panelRect);
  SetRect(&panelRect,204,284,222,299);
  EraseRect(&panelRect);
  SetRect(&panelRect,286,284,323,299);
  EraseRect(&panelRect);
  SetRect(&panelRect,389,284,416,299);
  EraseRect(&panelRect);
  SetRect(&panelRect,472,284,495,299);
  EraseRect(&panelRect);

  NumToString((SInt32) (*textEditStrucHdl)->teLength,textString);
  MoveTo(47,295);
  DrawString(textString);

  NumToString((SInt32) (*textEditStrucHdl)->nLines,textString);
  MoveTo(124,295);
  DrawString(textString);

  NumToString((SInt32) (*textEditStrucHdl)->lineHeight,textString);
  MoveTo(204,295);
  DrawString(textString);

  NumToString((SInt32) (*textEditStrucHdl)->destRect.top,textString);
  MoveTo(286,295);
  DrawString(textString);

  NumToString((SInt32) GetControlValue(controlRef),textString);
  MoveTo(389,295);
  DrawString(textString);

  NumToString((SInt32) GetControlMaximum(controlRef),textString);
  MoveTo(472,295);
  DrawString(textString);

  RGBForeColor(&blackColour);
  RGBBackColor(&whiteColour);
}

// ****************************************************************************** doErrorAlert

void  doErrorAlert(SInt16 errorCode)
{
  Str255 errorString;
  SInt16 itemHit;

  GetIndString(errorString,rErrorStrings,errorCode);

  if(errorCode < eWindow)
  {
    StandardAlert(kAlertStopAlert,errorString,NULL,NULL,&itemHit);
    ExitToShell();
  }
  else
  {
    StandardAlert(kAlertCautionAlert,errorString,NULL,NULL,&itemHit);
  }
}

// ************************************************************************** navEventFunction

void  navEventFunction(NavEventCallbackMessage callBackSelector,NavCBRecPtr callBackParms,
                       NavCallBackUserData callBackUD)
{
}

// *******************************************************************************************
// HelpDialog.c
// *******************************************************************************************
// ........................................................................................................ includes

#include <Carbon.h>

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

#define eHelpDialog           8
#define eHelpDocStructure     9
#define eHelpText             10
#define eHelpPicture          11
#define eHelpControls         12
#define rTextIntroduction     128
#define rTextCreatingText     129
#define rTextModifyHelp       130
#define rPictIntroductionBase 128
#define rPictCreatingTextBase 129
#define kTextInset            4

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

typedef struct
{
  Rect      bounds;
  PicHandle pictureHdl;
} pictInfoStructure;

typedef struct
{
  TEHandle          textEditStrucHdl;
  ControlRef        scrollbarHdl;
  SInt16            pictCount;
  pictInfoStructure *pictInfoStructurePtr;
}  docStructure, ** docStructureHandle;

typedef struct
{
  RGBColor     backColour;
  PixPatHandle backPixelPattern;
  Pattern      backBitPattern;
} backColourPattern;

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

GrafPtr                gOldPort;
EventHandlerUPP        helpWindowEventHandlerUPP;
ControlUserPaneDrawUPP userPaneDrawFunctionUPP;
ControlActionUPP       actionFunctionUPP;
SInt16                 gTextResourceID;
SInt16                 gPictResourceBaseID;
RgnHandle              gSavedClipRgn  = NULL;

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

void      doHelp                  (void);
OSStatus  helpWindowEventHandler  (EventHandlerCallRef,EventRef,void *);
void      userPaneDrawFunction    (ControlRef,SInt16);
Boolean   doGetText               (WindowRef,SInt16,Rect);
Boolean   doGetPictureInfo        (WindowRef,SInt16);
void      actionFunction          (ControlRef,SInt16);
void      doScrollTextAndPicts    (WindowRef);
void      doDrawPictures          (WindowRef,Rect *);
void      doCloseHelp             (WindowRef);
void      doDisposeDescriptors    (void);
void      doSetBackgroundWhite    (void);

extern void  doErrorAlert         (SInt16);

// ************************************************************************************ doHelp
```

```
void  doHelp(void)
{
  OSStatus           osError;
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  ControlRef         controlRef;
  ControlID          controlID;
  Rect               windowRect      = { 0,  0,  353,382 };
  Rect               pushButtonRect  = { 312,297,332,366 };
  Rect               userPaneRect    = { 16, 16, 296,351 };
  Rect               scrollBarRect   = { 16, 350,296,366 };
  Rect               popupButtonRect = { 312,12, 332,256 };
  Rect               destRect, viewRect;
  EventTypeSpec      dialogEvents[] = {{ kEventClassControl, kEventControlClick } };

  GetPort(&gOldPort);

  // ………………………………………………………………………………………………………………… create universal procedure pointers

  helpWindowEventHandlerUPP = NewEventHandlerUPP((EventHandlerProcPtr)
                                                 helpWindowEventHandler);
  userPaneDrawFunctionUPP = NewControlUserPaneDrawUPP((ControlUserPaneDrawProcPtr)
                                                       userPaneDrawFunction);
  actionFunctionUPP = NewControlActionUPP((ControlActionProcPtr) actionFunction);

  // ……………………………………………………………………………………………………………………………… create modal class window

  osError = CreateNewWindow(kMovableModalWindowClass,kWindowStandardHandlerAttribute,
                            &windowRect,&windowRef);
  if(osError == noErr)
  {
    RepositionWindow(windowRef,FrontWindow(),kWindowAlertPositionOnMainScreen);
    SetThemeWindowBackground(windowRef,kThemeBrushDialogBackgroundActive,false);

    InstallWindowEventHandler(windowRef,helpWindowEventHandlerUPP,
                          GetEventTypeCount(dialogEvents),dialogEvents,windowRef,NULL);

    if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
    {
      doErrorAlert(eHelpDocStructure);
      DisposeWindow(windowRef);
      doDisposeDescriptors();
      return;
    }

    SetWRefCon(windowRef,(SInt32) docStrucHdl);
    SetPortWindowPort(windowRef);

    // ……………………………………………… create root, push button, user pane, and scroll bar controls

    CreateRootControl(windowRef,&controlRef);

    if((osError = CreatePushButtonControl(windowRef,&pushButtonRect,CFSTR("OK"),&controlRef))
        == noErr)
    {
      SetWindowDefaultButton(windowRef,controlRef);
      controlID.id = 'done';
      SetControlID(controlRef,&controlID);
    }

    if(osError == noErr)
    {
      if((osError = CreateUserPaneControl(windowRef,&userPaneRect,0,&controlRef)) == noErr)
      {
        SetControlData(controlRef,kControlEntireControl,kControlUserPaneDrawProcTag,
                       sizeof(userPaneDrawFunctionUPP),(Ptr) &userPaneDrawFunctionUPP);
      }
    }
    if(osError == noErr)
    {
      if((osError = CreateScrollBarControl(windowRef,&scrollBarRect,0,0,1,0,true,
                                           actionFunctionUPP,&controlRef)) == noErr)
      (*docStrucHdl)->scrollbarHdl = controlRef;
      controlID.id = 'scro';
      SetControlID(controlRef,&controlID);
    }

    if(osError == noErr)
    {
      if((osError = CreatePopupButtonControl(windowRef,&popupButtonRect,CFSTR("Title:"),131,
                                             false,-1,0,0,&controlRef)) == noErr)
      controlID.id = 'popu';
      SetControlID(controlRef,&controlID);
    }

    if(osError != noErr)
    {
      doErrorAlert(eHelpControls);
      DisposeWindow(windowRef);
      doDisposeDescriptors();
      return;
    }
  }
  else
  {
    doErrorAlert(eHelpDialog);
    doDisposeDescriptors();
    return;
  }
  // .......................................... set destination and view rectangles, create TextEdit structure

  InsetRect(&userPaneRect,kTextInset,kTextInset / 2);
  destRect = viewRect = userPaneRect;
    (*docStrucHdl)->textEditStrucHdl = TEStyleNew(&destRect,&viewRect);

  // ................................ initialise picture information structure field of document structure

  (*docStrucHdl)->pictInfoStructurePtr = NULL;

  // ............................................ assign resource IDs of first topic's 'TEXT'/'styl' resources

  gTextResourceID      = rTextIntroduction;
  gPictResourceBaseID  = rPictIntroductionBase;

  // ........................................................ load text resources and insert into edit structure

  if(!(doGetText(windowRef,gTextResourceID,viewRect)))
  {
    doCloseHelp(windowRef);
    doDisposeDescriptors();
    return;
  }

  // ............ search for option-space charas in text and load same number of 'PICT' resources

  if(!(doGetPictureInfo(windowRef,gPictResourceBaseID)))
  {
    doCloseHelp(windowRef);
    doDisposeDescriptors();
    return;
  }

  // ............................................... create an empty region for saving the old clipping region

  gSavedClipRgn = NewRgn();

  // ........................................................................................ show window and run modal loop
```

```
  ShowWindow(windowRef);
  RunAppModalLoopForWindow(windowRef);
}

// ****************************************************************** helpWindowEventHandler

OSStatus  helpWindowEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                                 void *userData)
{
  OSStatus           result = eventNotHandledErr;
  WindowRef          windowRef;
  UInt32             eventClass;
  UInt32             eventKind;
  Point              mouseLocation;
  ControlRef         controlRef;
  ControlPartCode    controlPartCode;
  ControlID          controlID;
  MenuItemIndex      menuItem;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  Rect               viewRect;

  windowRef  = userData;
  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  if(eventClass == kEventClassControl)
  {
    if(eventKind == kEventControlClick)
    {
      GetEventParameter(eventRef,kEventParamMouseLocation,typeQDPoint,NULL,
                        sizeof(mouseLocation),NULL,&mouseLocation);
      GlobalToLocal(&mouseLocation);
      controlRef = FindControlUnderMouse(mouseLocation,windowRef,&controlPartCode);
      if(controlRef)
      {
        GetControlID(controlRef,&controlID);
        if(controlID.id == 'done')                                          // push button
        {
          if(TrackControl(controlRef,mouseLocation,NULL))
          {
            QuitAppModalLoopForWindow(windowRef);
            doCloseHelp(windowRef);
            doDisposeDescriptors();
            result = noErr;
          }
        }
        if(controlID.id == 'scro')                                           // scroll bar
        {
          TrackControl(controlRef,mouseLocation,actionFunctionUPP);
          result = noErr;
        }
        else if(controlID.id == 'popu')                               // pop-up menu button
        {
          TrackControl(controlRef,mouseLocation,(ControlActionUPP) -1);
          menuItem = GetControlValue(controlRef);
          switch(menuItem)
          {
            case 1:
              gTextResourceID     = rTextIntroduction;
              gPictResourceBaseID = rPictIntroductionBase;
              break;

            case 2:
              gTextResourceID     = rTextCreatingText;
              gPictResourceBaseID = rPictCreatingTextBase;
              break;
            case 3:
              gTextResourceID     = rTextModifyHelp;
              break;
          }

          docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
          textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
          viewRect = (*textEditStrucHdl)->viewRect;

          if(!(doGetText(windowRef,gTextResourceID,viewRect)))
          {
            doCloseHelp(windowRef);
            doDisposeDescriptors();
            return;
          }

          if(!(doGetPictureInfo(windowRef,gPictResourceBaseID)))
          {
            doCloseHelp(windowRef);
            doDisposeDescriptors();
            return;
          }

          doDrawPictures(windowRef,&viewRect);

          result = noErr;
        }
      }
    }
  }

  return result;
}

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

void  userPaneDrawFunction(ControlRef controlRef,SInt16 thePart)
{
  Rect               itemRect, viewRect;
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  Boolean           inState;

  windowRef = GetControlOwner(controlRef);

  GetControlBounds(controlRef,&itemRect);
  InsetRect(&itemRect,1,1);
  itemRect.right += 15;

  if(IsWindowVisible(windowRef))
    inState = IsWindowHilited(windowRef);
  DrawThemeListBoxFrame(&itemRect,inState);

  doSetBackgroundWhite();
  EraseRect(&itemRect);

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
  viewRect = (*textEditStrucHdl)->viewRect;

  TEUpdate(&viewRect,textEditStrucHdl);
  doDrawPictures(windowRef,&viewRect);
}

// ********************************************************************************* doGetText

Boolean  doGetText(WindowRef windowRef,SInt16 textResourceID,Rect viewRect)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  Handle             helpTextHdl;
  StScrpHandle       stylScrpStrucHdl;
  SInt16             numberOfLines, heightOfText, heightToScroll;

  doSetBackgroundWhite();

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  TESetSelect(0,32767,textEditStrucHdl);
  TEDelete(textEditStrucHdl);

  (*textEditStrucHdl)->destRect = (*textEditStrucHdl)->viewRect;
  SetControlValue((*docStrucHdl)->scrollbarHdl,0);

  helpTextHdl = GetResource('TEXT',textResourceID);
  if(helpTextHdl == NULL)
  {
    doErrorAlert(eHelpText);
    return false;
  }

  stylScrpStrucHdl = (StScrpHandle) GetResource('styl',textResourceID);
  if(stylScrpStrucHdl == NULL)
  {
    doErrorAlert(eHelpText);
    return false;
  }

  TEStyleInsert(*helpTextHdl,GetHandleSize(helpTextHdl),stylScrpStrucHdl,textEditStrucHdl);

  ReleaseResource(helpTextHdl);
  ReleaseResource((Handle) stylScrpStrucHdl);

  numberOfLines = (*textEditStrucHdl)->nLines;
  heightOfText = TEGetHeight((SInt32) numberOfLines,1,textEditStrucHdl);

  if(heightOfText > (viewRect.bottom - viewRect.top))
  {
    heightToScroll = TEGetHeight((SInt32) numberOfLines,1,textEditStrucHdl) -
                             (viewRect.bottom - viewRect.top);
    SetControlMaximum((*docStrucHdl)->scrollbarHdl,heightToScroll);
    ActivateControl((*docStrucHdl)->scrollbarHdl);
    SetControlViewSize((*docStrucHdl)->scrollbarHdl,(*textEditStrucHdl)->viewRect.bottom -
                    (*textEditStrucHdl)->viewRect.top);
  }
  else
  {
    DeactivateControl((*docStrucHdl)->scrollbarHdl);
  }

  return true;
}

// ************************************************************************** doGetPictureInfo

Boolean  doGetPictureInfo(WindowRef windowRef,SInt16 firstPictID)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  Handle             textHdl;
  SInt32             offset, textSize;
  SInt16             numberOfPicts, a, lineHeight, fontAscent;
  SInt8              optionSpace[1] = "\xCA";
  pictInfoStructure  *pictInfoPtr;
  Point              picturePoint;
  TextStyle          whatStyle;
```

```
docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

if((*docStrucHdl)->pictInfoStructurePtr != NULL)
{
  for(a=0;a<(*docStrucHdl)->pictCount;a++)
    ReleaseResource((Handle) (*docStrucHdl)->pictInfoStructurePtr[a].pictureHdl);

  DisposePtr((Ptr) (*docStrucHdl)->pictInfoStructurePtr);
  (*docStrucHdl)->pictInfoStructurePtr = NULL;
}

(*docStrucHdl)->pictCount = 0;

textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
textHdl = (*textEditStrucHdl)->hText;

textSize = GetHandleSize(textHdl);
offset = 0;
numberOfPicts = 0;

HLock(textHdl);

offset = Munger(textHdl,offset,optionSpace,1,NULL,0);
while((offset >= 0) && (offset <= textSize))
{
  numberOfPicts++;
  offset++;
  offset = Munger(textHdl,offset,optionSpace,1,NULL,0);
}

if(numberOfPicts == 0)
{
  HUnlock(textHdl);
  return true;
}

pictInfoPtr = (pictInfoStructure *) NewPtr(sizeof(pictInfoStructure) * numberOfPicts);
(*docStrucHdl)->pictInfoStructurePtr = pictInfoPtr;

offset = 0L;

for(a=0;a<numberOfPicts;a++)
{
  pictInfoPtr[a].pictureHdl = GetPicture(firstPictID + a);
  if(pictInfoPtr[a].pictureHdl == NULL)
  {
    doErrorAlert(eHelpPicture);
    return false;
  }

  offset = Munger(textHdl,offset,optionSpace,1,NULL,0);
  picturePoint = TEGetPoint((SInt16)offset,textEditStrucHdl);

  TEGetStyle(offset,&whatStyle,&lineHeight,&fontAscent,textEditStrucHdl);
  picturePoint.v -= lineHeight;
  offset++;
  pictInfoPtr[a].bounds = (**pictInfoPtr[a].pictureHdl).picFrame;

  OffsetRect(&pictInfoPtr[a].bounds,
             (((*textEditStrucHdl)->destRect.right + (*textEditStrucHdl)->destRect.left) -
             (pictInfoPtr[a].bounds.right + pictInfoPtr[a].bounds.left) ) / 2,
             - pictInfoPtr[a].bounds.top + picturePoint.v);
}

(*docStrucHdl)->pictCount = a;

HUnlock(textHdl);
```

```
  return true;
}

// ************************************************************************ actionFunction

void  actionFunction(ControlRef scrollbarHdl,SInt16 partCode)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             delta, oldValue, offset, lineHeight, fontAscent;
  Point              thePoint;
  Rect               viewRect, portRect;
  TextStyle          style;

  if(partCode)
  {
    windowRef = GetControlOwner(scrollbarHdl);
    docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
    textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
    viewRect = (*textEditStrucHdl)->viewRect;
    thePoint.h = viewRect.left + kTextInset;

    if(partCode != kControlIndicatorPart)
    {
      switch(partCode)
      {
        case kControlUpButtonPart:
          thePoint.v = viewRect.top - 4;
          offset = TEGetOffset(thePoint,textEditStrucHdl);
          thePoint = TEGetPoint(offset,textEditStrucHdl);
          TEGetStyle(offset,&style,&lineHeight,&fontAscent,textEditStrucHdl);
          delta = thePoint.v - lineHeight - viewRect.top;
          break;

        case kControlDownButtonPart:
          thePoint.v = viewRect.bottom + 2;
          offset = TEGetOffset(thePoint,textEditStrucHdl);
          thePoint = TEGetPoint(offset,textEditStrucHdl);
          delta = thePoint.v - viewRect.bottom;
          break;

        case kControlPageUpPart:
          thePoint.v = viewRect.top + 2;
          offset = TEGetOffset(thePoint,textEditStrucHdl);
          thePoint = TEGetPoint(offset,textEditStrucHdl);
          TEGetStyle(offset,&style,&lineHeight,&fontAscent,textEditStrucHdl);
          thePoint.v += lineHeight - fontAscent;
          thePoint.v -= viewRect.bottom - viewRect.top;
          offset = TEGetOffset(thePoint,textEditStrucHdl);
          thePoint = TEGetPoint(offset,textEditStrucHdl);
          TEGetStyle(offset,&style,&lineHeight,&fontAscent,textEditStrucHdl);
          delta = thePoint.v - viewRect.top;
          if(offset == 0)
            delta -= lineHeight;
          break;

        case kControlPageDownPart:
          thePoint.v = viewRect.bottom - 2;
          offset = TEGetOffset(thePoint,textEditStrucHdl);
          thePoint = TEGetPoint(offset,textEditStrucHdl);
          TEGetStyle(offset,&style,&lineHeight,&fontAscent,textEditStrucHdl);
          thePoint.v -= fontAscent;
          thePoint.v += viewRect.bottom - viewRect.top;
          offset = TEGetOffset(thePoint,textEditStrucHdl);
          thePoint = TEGetPoint(offset,textEditStrucHdl);
          TEGetStyle(offset,&style,&lineHeight,&fontAscent,textEditStrucHdl);
          delta =  thePoint.v - lineHeight - viewRect.bottom;
          if(offset == (**textEditStrucHdl).teLength)
```

```
            delta += lineHeight;
          break;
      }

      oldValue = GetControlValue(scrollbarHdl);

      if(((delta < 0) && (oldValue > 0)) || ((delta > 0) &&
         (oldValue < GetControlMaximum(scrollbarHdl))))
      {
        GetClip(gSavedClipRgn);
        GetWindowPortBounds(windowRef,&portRect);
        ClipRect(&portRect);

        SetControlValue(scrollbarHdl,oldValue + delta);
        SetClip(gSavedClipRgn);
      }
    }

    doScrollTextAndPicts(windowRef);
  }
}

// ********************************************************************* doScrollTextAndPicts

void  doScrollTextAndPicts(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             scrollDistance, oldScroll;
  Rect               updateRect;

  doSetBackgroundWhite();

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  oldScroll = (*textEditStrucHdl)->viewRect.top -(*textEditStrucHdl)->destRect.top;
  scrollDistance = oldScroll - GetControlValue((*docStrucHdl)->scrollbarHdl);
  if(scrollDistance == 0)
    return;

  TEScroll(0,scrollDistance,textEditStrucHdl);

  if((*docStrucHdl)->pictCount == 0)
    return;

  updateRect = (*textEditStrucHdl)->viewRect;

  if(scrollDistance > 0)
  {
    if(scrollDistance < (updateRect.bottom - updateRect.top))
      updateRect.bottom = updateRect.top + scrollDistance;
  }
  else
  {
    if( - scrollDistance < (updateRect.bottom - updateRect.top))
      updateRect.top = updateRect.bottom + scrollDistance;
  }

  doDrawPictures(windowRef,&updateRect);
}

// *************************************************************************** doDrawPictures

void  doDrawPictures(WindowRef windowRef,Rect *updateRect)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             pictCount, pictIndex, vOffset;
  PicHandle          thePictHdl;
  Rect               pictLocRect, dummyRect;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  vOffset = (*textEditStrucHdl)->destRect.top -
            (*textEditStrucHdl)->viewRect.top - kTextInset;
  pictCount = (*docStrucHdl)->pictCount;

  for(pictIndex = 0;pictIndex < pictCount;pictIndex++)
  {
    pictLocRect = (*docStrucHdl)->pictInfoStructurePtr[pictIndex].bounds;
    OffsetRect(&pictLocRect,0,vOffset);

    if(!SectRect(&pictLocRect,updateRect,&dummyRect))
      continue;

    thePictHdl = (*docStrucHdl)->pictInfoStructurePtr[pictIndex].pictureHdl;

    LoadResource((Handle) thePictHdl);
    HLock((Handle) thePictHdl);

    GetClip(gSavedClipRgn);
    ClipRect(updateRect);
    DrawPicture(thePictHdl,&pictLocRect);

    SetClip(gSavedClipRgn);
    HUnlock((Handle) thePictHdl);
  }
}

// ****************************************************************************** doCloseHelp

void  doCloseHelp(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             a;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  if(gSavedClipRgn)
    DisposeRgn(gSavedClipRgn);

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

  if((*docStrucHdl)->pictInfoStructurePtr)
  {
    for(a=0;a<(*docStrucHdl)->pictCount;a++)
      ReleaseResource((Handle) (*docStrucHdl)->pictInfoStructurePtr[a].pictureHdl);
    DisposePtr((Ptr) (*docStrucHdl)->pictInfoStructurePtr);
  }

  DisposeHandle((Handle) docStrucHdl);
  DisposeWindow(windowRef);
  SetPort(gOldPort);
}

// ********************************************************************** doDisposeDescriptors

void  doDisposeDescriptors(void)
{
  DisposeEventHandlerUPP(helpWindowEventHandlerUPP);
  DisposeControlUserPaneDrawUPP(userPaneDrawFunctionUPP);
  DisposeControlActionUPP(actionFunctionUPP);
}
```

```
// ********************************************************************** doSetBackgroundWhite

void  doSetBackgroundWhite(void)
{
  RGBColor whiteColour = { 0xFFFF, 0xFFFF, 0xFFFF };
  Pattern  whitePattern;

  RGBBackColor(&whiteColour);
  BackPat(GetQDGlobalsWhite(&whitePattern));
}

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

Demonstration Program MonoTextEdit Comments

When this program is run, the user should explore both the text editor and the Help dialog.

Text Editor

In the text editor, the user should perform all the actions usually associated with a simple text editor, that is:

- Open a new document window, open an existing 'TEXT' file for display in a new document window, and save a document to a 'TEXT' file. (A 'TEXT' file titled "MonoTextEdit Document" is included.)

- Enter new text and use the Edit menu Cut, Copy, Paste, and Clear commands to edit the text. (Pasting between documents and other applications is supported.)

- Select text by clicking and dragging, double-clicking a word, shift-clicking, and choosing the Select All command from the Edit menu. Also select large amounts of text by clicking in the text and dragging the cursor above or below the window so as to invoke auto-scrolling.

- Scroll a large document by dragging the scroll box/scroller (live scrolling is used), clicking once in a scroll arrow or gray area/track, and holding the mouse down in a scroll arrow or gray area/track.

Whenever any action is taken, the user should observe the changes to the values displayed in the data panel at the bottom of each window. In particular, the relationship between the destination rectangle and scroll bar control value should be noted.

The user should also note that outline highlighting is activated for all windows and that the forward-delete key is supported by the application. (The forward-delete key is not supported by TextEdit.)

Help Dialog

The user should choose MonoTextEdit Help from the Mac OS 8/9 Apple menu or Mac OS X Help menu to open the Help dialog and then scroll through the three help topics, which may be chosen in the pop-up menu at the bottom of the dialog. The help topics contain documentation on the Help dialog which supplements the source code comments below.

MonoTextEdit.c

defines

kMaxTELength represents the maximum allowable number of bytes in a TextEdit structure. kTab, kBackSpace, kForwardDelete, kReturn, and kEscape representing the character codes generated by the tab, delete, forward delete, Return and escape keys.

typedefs

The docStructure data type will be used for a small document structure comprising a handle to a TextEdit structure and a handle to a vertical scroll bar.

Global Variables

scrollActionFunctionUPP will be assigned a universal procedure pointer to an action (callback) function for the scroll bar. customClickLoopUPP will be assigned a universal procedure pointer to a custom click loop (callback) function. gOldControlValue will be assigned the scroll bar's control value.

main

The main function creates universal procedure pointers for the application-defined scroll action and custom click loop (callback) functions and installs a timer set to fire repeatedly at the interval returned by a call to GetCaretTime. When the timer fires, the function doIdle is called.

windowEventHandler

When the kEventClassMouse event type is received, if the call to FindControlUnderMouse reveals that a control (that is, the vertical scroll bar) is under the mouse, TrackControl is called with a universal procedure pointer to an action function passed in the third parameter.

When the kEventRawKeyDown and kEventRawKeyRepeat event types are received, the function doKeyEvent is called only if the Command key was not down.

doIdle

doIdle is called whenever the installed timer fires.

The first line gets a reference to the front window. If the front window is a document window, a handle to the window's document structure is retrieved. A handle to the TextEdit structure associated with the window is stored in the document structure's textEditStrucHdl field. This is passed in the call to TEIdle, which blinks the insertion point caret.

doKeyEvent

doKeyEvent is called when the kEventRawKeyDown and kEventRawKeyRepeat event types are received. It handles all key-down events that are not Command key equivalents.

If the character code is equal to that for the escape key or lower, and except if it is the carriage return or backspace character, the function simply returns.

The first three lines get a handle to the TextEdit structure whose handle is stored in the front window's document structure.

The next line filters out the tab key character code. (TextEdit does not support the tab key and some applications may need to provide a tab key handler.)

The next character code to be filtered out is the forward-delete key character code. TextEdit does not recognise this key, so this else if block provides forward-delete key support for the program. The first line in this block gets the current selection length from the TextEdit structure. If this is zero (that is, there is no selection range and an insertion point is being displayed), the selEnd field is incremented by one. This, in effect, creates a selection range comprising the character following the insertion point. TEDelete deletes the current selection range from the TextEdit structure. Such deletions could change the number of text lines in the TextEdit structure, requiring the vertical scroll bar to be adjusted; hence the call to the function doAdjustScrollbar.

Processing of those character codes which have not been filtered out is performed in the else block. A new character must not be allowed to be inserted if the TextEdit limit of 32,767 characters will be exceeded. Accordingly, and given that TEKey replaces the selection range with the character passed to it, the first step is to get the current selection length. If the current number of characters minus the selection length plus 1 is less than 32,767, the character code is passed to TEKey for insertion into the TextEdit structure. In addition, and since all this could change the number of lines in the TextEdit structure, the scroll bar adjustment function is called.

If the TextEdit limit will be exceeded by accepting the character, an alert is invoked advising the user of the situation.

The last line calls a function which prints data extracted from the edit text and control structures at the bottom of the window.

scrollActionFunction

scrollActionFunction is associated with the vertical scroll bar. It is the callback function which will be repeatedly called by TrackControl when the kEventMouseDown event type is received. It will be called repeatedly while the mouse button remains down in the scroll box/scroller, scroll arrows or gray areas/track of the vertical scroll bar.

The first line gets a reference to the window object for the window which "owns" the control. The next two lines get a handle to the TextEdit structure associated with the window.

Within the outer if block, the first if block executes if the control part is not the scroll box/scroller (that is, the indicator). The purpose of the switch is to get a value into the variable linesToScroll. If the mouse-down was in a scroll arrow, that value will be 1. If the mouse-down was in a gray area/track, that value will be equivalent to one less than the number of text lines that will fit in the view rectangle. (Subtracting 1 from the total number of lines that will fit in the view rectangle ensures that the line of text at the bottom/top of the view rectangle prior to a gray area/track scroll will be visible at the top/bottom of the window after the scroll.)

Immediately after the switch, the value in linesToScroll is changed to a negative value if the mouse-down occurred in either the down scroll arrow or down gray area/track.

The next block ensures that no scrolling action will occur if the document is currently scrolled fully up (control value equals control maximum) or fully down (control value equals 0). In either case, linesToScroll will be set to 0, meaning that the call to TEScroll near the end of the function will not occur.

SetControlValue sets the control value to the value just previously calculated, that is, to the current control value minus the value in linesToScroll.

The next line sets the value in linesToScroll back to what it was before the line linesToScroll = controlvalue - linesToScroll executed. This value, multiplied by the value in the lineHeight field of the TextEdit structure, is later passed to TEScroll as the parameter which specifies the number of pixels to scroll.

If the control part is the scroll box/scroller (that is, the indicator), the variable linesToScroll is assigned a value equal to the control's value as it was the last time this function was called minus the control's current value. The global variable which holds the control's "old" value is then assigned the control's current value preparatory to the next call to this function.

With the number of lines to scroll determined, TEScroll is called to scroll the text within the view rectangle by the number of pixels specified in the second parameter. (Positive values scroll the text towards the bottom of the screen. Negative values scroll the text towards the top.)

The last line is for demonstration purposes only. It calls the function which prints data extracted from the edit and control structures at the bottom of the window.

doInContent

doInContent is called when the kEventWindowClickContentRgn event type is received.

The first three lines retrieve a handle to the TextEdit structure associated with the front window. The call to PtInRect checks whether the mouse-down occurred within the view rectangle. (Note that the view rectangle is in local coordinates, so the mouse-down coordinates passed as the first parameter to the PtInRect call must also be in local coordinates.) If the mouse-down was in the view rectangle, TEClick is called to advise TextEdit of the mouse-down event. Note that the position of the shift key is passed in the second parameter. (TEClick's behaviour depends on the position of the shift key.)

doDrawContent

doDrawContent is called when the kEventWindowDrawContent event type is received.

The first two lines get the handle to the TextEdit structure associated with the window.

UpdateControls is called to draw the scroll bar. The call to TEUpdate draws the text currently in the TextEdit structure.

doActivateDeactivate

doActivateDeactivatew performs window activation/deactivation. It is called when the kEventWindowActivated and kEventWindowDecativated event types are received.

The first two lines retrieve a handle to the TextEdit structure for the window. If the window is becoming active, its graphics port is set as the current graphics port. The bottom of the view rectangle is then adjusted so that the height of the view rectangle is an exact multiple of the value in the lineHeight field of the TextEdit structure. (This avoids the possibility of only part of the full height of a line of text appearing at the bottom of the view rectangle.) TEActivate activates the TextEdit structure associated with the window, ActivateControl activates the scroll bar, doAdjustScrollbar adjusts the scroll bar, and doAdjustCursor adjusts the cursor shape.

If the window is becoming inactive, TEDeactivate deactivates the TextEdit structure associated with the window and DeactivateControl deactivates the scroll bar.

doNewDocWindow

doNewDocWindow is called at program launch and when the user chooses New or Open from the File menu. It opens a new window, associates a document structure with that window, creates a vertical scroll bar, creates a monostyled TextEdit structure, installs the custom click loop (callback) function, enables automatic scrolling, and enables outline highlighting.

The call to CreateNewWindow and the following block creates a new window with the standard document window attributes less the size box/resize control. Note that the window's graphics port is set as the current port before the later call to TENew. (Since the TextEdit structure assumes the drawing environment specified in the graphics port structure, setting the graphics port must be done before TENew creates the TextEdit structure.)

The call to TextSize sets the text size. This, together with the default application font, will be copied from the graphics port to the TextEdit structure when TENew is called.)

After the window event handler is installed, a document structure is created and the handle stored in the window's window object. The following line increments the global variable which keeps track of the number of open windows. GetNewControl creates a vertical scroll bar and assigns a handle to it to the appropriate field of the document structure. The next block establishes the view and destination rectangles two pixels inside the window's port rectangle less the scroll bar.

MoveHHi and HLock move the document structure high and lock it. A monostyled TextEdit structure is then created by TENew and its handle is assigned to the appropriate field of the document structure. (If this call is not successful, the window and scroll bar are disposed of, an error alert is displayed, and the function returns.) The handle to the document structure is then unlocked.

TESetClickLoop installs the universal procedure pointer to the custom click loop (callback) function customClickLoop in the clickLoop field of the TextEdit structure. TEAutoView enables automatic scrolling for the TextEdit structure. TEFeatureFlag enables outline highlighting for the TextEdit structure.

The last line returns a reference to the newly opened window's window object.

customClickLoop

customClickLoop replaces the default click loop function so as to provide for scroll bar adjustment in concert with automatic scrolling. Following a mouse-down within the view rectangle, customClickLoop is called repeatedly by TEClick as long as the mouse button remains down.

The first three lines retrieve a handle to the TextEdit structure associated with the window. The next two lines save the current graphics port and set the window's graphics port as the current port.

The window's current clip region will have been set by TextEdit to be equivalent to the view rectangle. Since the scroll bar has to be redrawn, the clipping region must be temporarily reset to include the scroll bar. Accordingly, GetClip saves the current clipping region and the following two lines set the clipping region to the bounds of the coordinate plane.

GetMouse gets the current position of the cursor. If the cursor is above the top of the port rectangle, the text must be scrolled downwards. Accordingly, the variable linesToScroll is set to 1. The subsidiary function doSetScrollBarValue (see below) is then called to, amongst other things, reset the scroll bar's value. Note that the value in linesToScroll may be modified by doSetScrollBarValue. If linesToScroll is not set to 0 by doSetScrollBarValue, TEScroll is called to scroll the text by a number of pixels equivalent to the value in the lineHeight field of the TextEdit structure, and in a downwards direction.

If the cursor is below the bottom of the port rectangle, the same process occurs except that the variable linesToScroll is set to -1, thus causing an upwards scroll of the text (assuming that the value in linesToScroll is not changed to 0 by doSetScrollBarValue).

If scrolling has occurred, doDrawDataPanel redraws the data panel. SetClip restores the clipping region to that established by the view rectangle and SetPort restores the saved graphics port. Finally, the last line returns true. (A return of false would cause TextEdit to stop calling customClickLoop, as if the user had released the mouse button.)

doSetScrollBarValue

doSetScrollBarValue is called from customClickLoop. Apart from setting the scroll bar's value so as to cause the scroll box to follow up automatic scrolling, the function checks whether the limits of scrolling have been reached.

The first two lines get the current control value and the current control maximum value. At the next block, the value in the variable linesToScroll will be set to either 0 (if the current control value is 0) or equivalent to the control maximum value (if the current control value is equivalent to the control maximum value. If these modifications do not occur, the value in linesToScroll will remain as established at the first line in this block, that is, the current control value minus the value in linesToScroll as passed to the function.

SetControlValue sets the control's value to the value in linesToScroll. The last line sets the value in linesToScroll to 0 if the limits of scrolling have already been reached, or to the value as it was when the doSetScrollBarValue function was entered.

doAdjustMenus

doAdjustMenus adjusts the menus. Much depends on whether any windows are currently open.

If at least one window is open, the first three lines in the if block get a handle to the TextEdit structure associated with the front window and the first call to EnableMenuItem enables the Close item in the File menu. If there is a current selection range, the Cut, Copy, and Clear items are enabled, otherwise they are disabled. If there is data of flavour type 'TEXT' in the scrap (the call to

GetScrapFlavourFlags), the Paste item is enabled, otherwise it is disabled. If there is any text in the TextEdit structure, the SaveAs and Select All items are enabled, otherwise they are disabled.

If no windows are open, the Close, SaveAs, Clear, and Select All items are disabled.

doMenuChoice

If the MonoTextEdit Help item in the Mac OS 8/9 Apple menu or Mac OS X Help menu is chosen, the function doHelp is called.

doFileMenu

doFileMenu handles File menu choices, calling the appropriate functions according to the menu item chosen. In the SaveAs case, a handle to the TextEdit structure associated with the front window is retrieved and passed as a parameter to the function doSaveAsFile.

Note that, because TextEdit, rather than file operations, is the real focus of this program, the file-related code has been kept to a minimum, even to the extent of having no Save-related, as opposed to SaveAs-related, code.

doEditMenu

doEditMenu handles choices from the Edit menu. Recall that, in the case of monostyled TextEdit structures, TECut, TECopy, and TEPaste do not copy/paste text to/from the scrap. This program, however, supports copying/pasting to/from the scrap.

Before the usual switch is entered, a handle to the TextEdit structure associated with the front window is retrieved.

The iCut case handles the Cut command. Firstly, the call to ClearCurrentScrap attempts to clear the scrap. If the call succeeds, PurgeSpace establishes the size of the largest block in the heap that would be available if a general purge were to occur. The next line gets the current selection length. If the selection length is greater than the available memory, the user is advised via an error message. Otherwise, TECut is called to remove the selected text from the TextEdit structure and copy it to the TextEdit private scrap. The scroll bar is adjusted, and TEToScrap is called to copy the private scrap to the scrap. If the TEToScrap call is not successful, ClearCurrentScrap cleans up as best it can by emptying the scrap.

The iCopy case handles the Copy command. If the call to ClearCurrentScrap to empty the scrap is successful, TECopy is called to copy the selected text from the TextEdit structure to the TextEdit private scrap. TEToScrap then copies the private scrap to the scrap. If the TEToScrap call is not successful, ClearCurrentScrap cleans up as best it can by emptying the scrap.

The iPaste case handles the Paste command, which must not proceed if the paste would cause the TextEdit limit of 32,767 bytes to be exceeded. The third line establishes a value equal to the number of bytes in the TextEdit structure plus the number of bytes of the 'TEXT' flavour type in the scrap. If this value exceeds the TextEdit limit, the user is advised via an error message. Otherwise, TEFromScrap copies the scrap to TextEdit's private scrap, TEPaste inserts the private scrap into the TextEdit structure, and the following line adjusts the scroll bar.

The iClear case handles the Clear command. TEDelete deletes the current selection range from the TextEdit structure and the following line adjusts the scroll bar.

The iSelectAll case handle the Select All command. TESetSelect sets the selection range according to the first two parameters (selStart and selEnd).

doGetSelectLength

doGetSelectLength returns a value equal to the length of the current selection.

doAdjustScrollbar

doAdjustScrollbar adjusts the vertical scroll bar.

The first two lines retrieve handles to the document structure and TextEdit structure associated with the window in question.

At the next block, the value in the nLines field of the TextEdit structure is assigned to the numberOfLines variable. The next action is somewhat of a refinement and is therefore not essential. If the last character in the TextEdit structure is the return character, numberOfLines is incremented by one. This will ensure that, when the document is scrolled to its end, a single blank line will appear below the last line of text.

At the next block, the variable controlMax is assigned a value equal to the number of lines in the TextEdit structure less the number of lines that will fit in the view rectangle. If this value is less than 0 (indicating that the number of lines in the TextEdit structure is less than the number of lines that will fit in the view rectangle), controlMax is set to 0. SetControlMaximum then sets the control maximum value. If controlMax is 0, the scroll bar is automatically unhighlighted by the SetControlMaximum call.

The first line of the next block assigns to the variable controlValue a value equal to the number of text lines that the top of the destination rectangle is currently "above" the top of the view rectangle. If the calculation returns a value less than 0 (that is, the document has been scrolled fully down), controlValue is set to 0. If the calculation returns a value greater than the current control maximum value (that is, the document has been scrolled fully up), controlValue is set to equal that value. SetControlValue sets the control value to the value in controlValue. For example, if the top of the view rectangle is 2, the top of the destination rectangle is -34 and the lineHeight field of the TextEdit structure contains the value 13, the control value will be set to 3.

SetControlViewSize is called to advise the Control Manager of the height of the view rectangle. This will cause the scroll box/scroller to be a proportional scroll box/scroller. (On Mac OS 8/9, this assumes that the user has selected Smart Scrolling on in the Appearance control panel.)

With the control maximum value and the control value set, TEScroll is called to make sure the text is scrolled to the position indicated by the scroll box/scroller. Extending the example in the previous paragraph, the second parameter in the TEScroll call is 2 - (34 - (3 * 13)), that is, 0. In that case, no corrective scrolling actually occurs.

doAdjustCursor

doAdjustCursor is called when the kEventMouseMoved, and kEventWindowActivated event types are received. It adjusts the cursor to the I-Beam shape when the cursor is over the content region less the scroll bar area, and to the arrow shape when the cursor is outside that region. It is similar to the cursor adjustment function in the demonstration program GworldPicCursIcn (Chapter 13).

doCloseWindow

doCloseWindow is called when the kEventWindowClose event type is received and when the user chooses Close from the File menu. It disposes of the specified window. The associated scroll bar, the associated TextEdit structure and the associated document structure are disposed of before the call to DisposeWindow.

doSaveAsFile, doOpenCommand, doOpenFile

The functions doSaveAsFile, doOpenCommand, and doOpenFile are document saving and opening functions, enabling the user to open and save 'TEXT' documents. Since the real focus of this program is TextEdit, not file operations, the code is "bare bones" and as brief as possible, Navigation Services 2.0 functions being used rather than the Navigation Services 3.0 functions used in the demonstration program Files (Chapter 18).

> For a complete example of opening and saving monostyled 'TEXT' documents, see the demonstration program at Files (Chapter 18).

doDrawDataPanel

doDrawDataPanel draws the data panel at the bottom of each window. Displayed in this panel are the values in the teLength, nLines, lineHeight and destRect.top fields of the TextEdit structure and the contrlValue and contrlMax fields of the scroll bar's control structure.

HelpDialog.c

defines

Constants are established for the index of error strings within a 'STR#' resource and 'TEXT', 'styl', and 'PICT' resource IDs. kTextInset which will be used to inset the view and destination rectangles a few pixels inside a user pane's rectangle.

typedef

The first two data types are for a picture information structure and a document structure. (Note that one field in the document structure is a pointer to a picture information structure.) The third data type will be used for saving and restoring the background colour and pattern.

doHelp

doHelp is called when the user chooses the MonoTextEdit Help item in the Help menu.

The dialog will utilise a window event handler, a user pane drawing (callback) function, and a control action (callback) function. The first block creates the associated universal procedure pointers.

The call to CreateNewWindow creates a new window of the movable modal class. NewHandle creates a block for a document structure and the handle is stored in the window object.

The next block creates the dialog's controls and assigns an ID to each. In the case of the user pane control, SetControlData is called to set a user pane drawing function. In the case of the scroll bar, the control reference is assigned to the appropriate field of the dialog's document structure.

At the next block, the destination and view rectangles are both made equal to the user pane's rectangle, but inset four pixels from the left and right and two pixels from the top and bottom. The call to TEStyleNew creates a multistyled TextEdit structure based on those two rectangles.

A pointer to a picture information structure will eventually be assigned to a field in the document structure. For the moment, that field is set to NULL.

At the next block, two global variables are assigned the resource IDs relating to the first Help topic's 'TEXT'/'styl' resource and associated 'PICT' resources.

The next block calls the function doGetText which, amongst other things, loads the specified 'TEXT'/'styl' resources and inserts the text and style information into the TextEdit structure.

The next block calls the function doGetPictureInfo which, amongst other things, searches for option-space characters in the 'TEXT' resource and, if option-space characters are found, loads a like number of 'PICT' resources beginning with the specified ID.

NewRgn creates an empty region, which will be used to save the dialog's graphic's port's clipping region.

To complete the initial setting up, ShowWindow is called to makes the dialog visible, following which RunAppModalLoopForWindow is called to run the nodal loop.

helpWindowEventHandler

helpWindowEventHandler is the event handler for the dialog. It responds to mouse clicks in the dialog's three controls.

If the push button was clicked, QuitAppModalLoopForWindow is called to terminate the modal loop and restore menu activation/deactivation status to that which obtained prior to the call to RunAppModalLoopForWindow, and the dialog is closed down.

Note that, if the click was in the scroll bar, TrackControl is called with a universal procedure pointer to an application-defined action (callback) function is passed in the actionProc parameter.

If the click was in the pop-up menu button, the menu item chosen is determined, and the switch assigns the appropriate 'TEXT'/'styl' and 'PICT' resource IDs to the global variables which keep track of which of those resources are to be loaded and displayed.

The next three lines get the view rectangle from the TextEdit structure, allowing the next blocks to perform the same "get text" and "get picture information" actions as were preformed at start-up, but this time with the 'TEXT'/'styl' and 'PICT' resources as determined within the preceding switch.

The call to doDrawPictures draws any pictures that might initially be located in the view rectangle.

userPaneDrawFunction

userPaneDrawFunction is the user pane drawing function set within doHelp.

The first line gets a reference to the user pane control's owning window. The next three lines get the user pane's rectangle, insets that rectangle by one pixel all round, and then further expands it to the right edge of the scroll bar. At the next block, a list box frame is drawn in the appropriate state, depending on whether the the movable modal dialog is currently the active window.

The next block erases the previously defined rectangle with the white colour using the white pattern.

The next three lines retrieve the view rectangle from the TextEdit structure. The call to TEUpdate draws the text in the TextEdit structure in the view rectangle. The call to doDrawPictures draws any pictures that might currently be located in the view rectangle.

doGetText

doGetText is called when the dialog is first opened and when the user chooses a new item from the pop-up menu. Amongst other things, it loads the 'TEXT'/'styl' resources associated with the current menu item and inserts the text and style information into the TextEdit structure.

The first two lines get a handle to the TextEdit structure. The next two lines set the selection range to the maximum value and then delete that selection. The destination rectangle is then made equal to the view rectangle and the scroll bar's value is set to 0.

GetResource is called twice to load the specified 'TEXT'/'styl' resources, following which TEStyleInsert is called to insert the text and style information into the TextEdit structure. Two calls to ReleaseResource then release the 'TEXT'/'styl' resources.

The next block gets the total height of the text in pixels.

At the next block, if the height of the text is greater than the height of the view rectangle, the local variable heightToScroll is made equal to the total height of the text minus the height of the view rectangle. This value is then used to set the scroll bar's maximum value. The scroll bar is then made active.

SetControlViewSize is called to advise the Control Manager of the height of the view rectangle. This will cause the scroll box to be a proportional scroll box.

If the height of the text is less than the height of the view rectangle, the scroll bar is made inactive.

true is returned if the GetResource calls did not return with false.

doGetPictureInfo

doGetPictureInfo is called after getText when the dialog is opened and when the user chooses a new item from the pop-up menu. Amongst other things, it searches for option-space characters in the 'TEXT' resource and, if option-space characters are found, loads a like number of 'PICT' resources beginning with the specified ID.

The first line gets a handle to the dialog's document structure.

If the picInfoRecPtr field of the document structure does not contain NULL, the currently loaded 'PICT' resources are released, the picture information structures are disposed of, and the picInfoRecPtr field of the document structure is set to NULL.

The next line sets to 0 the field of the document structure which keeps track of the number of pictures associated with the current 'TEXT' resource.

The next two lines get a handle to the TextEdit structure, then a handle to the block containing the actual text. This latter is then used to assign the size of that block to a local variable. After two local variables are initialised, the block containing the text is locked.

The next block counts the number of option-space characters in the text block. At the following block, if there are no option-space characters in the block, the block is unlocked and the function returns.

A call to NewPtr then allocates a nonrelocatable block large enough to accommodate a number of picture information structures equal to the number of option-space characters found. The pointer to the block is then assigned to the appropriate field of the dialog's document structure.

The next line resets the offset value to 0.

The for loop repeats for each of the option-space characters found. GetPicture loads the specified 'PICT' resource (the resource ID being incremented from the base ID at each pass through the loop) and assigns the handle to the appropriate field of the relevant picture information structure. Munger finds the offset to the next option-space character and TEGetPoint gets the point, based on the destination rectangle, of the bottom left of the character at that offset. TEGetStyle is called to obtain the line height of the character at the offset and this value is subtracted from the value in the point's v field. The offset is incremented and the rectangle in the picture structure's picFrame field is assigned to the bounds field of the picture information structure. The next block then offsets this rectangle so that it is centred laterally in the destination rectangle with its top offset from the top of the destination rectangle by the amount established at the line picturePoint.v -= lineHeight;.

The third last line assigns the number of pictures loaded to the appropriate field of the dialog's document structure. The block containing the text is then unlocked. The function returns true if false has not previously been returned within the for loop.

actionFunction

actionFunction is the action function called from within the event filter (callback) function eventFilter. It is repeatedly called by TrackControl while the mouse button remains down within the scroll bar. Its ultimate purpose is to determine the new scrollbar value when the mouse-down is within the scroll arrows or gray areas/track of the scroll bar, and then call a separate function to effect the actual scrolling of the text and pictures based on the new scrollbar value. (The scroll bar is the live scrolling variant, so the CEDF automatically updates the control's value while the mouse remains down in the scroll box/scroller.)

Firstly, if the cursor is still not within the control, execution falls through to the bottom of the function and the action function exits.

The first block gets a pointer to the owner of the scrollbar, retrieves a handle to the dialog's document structure, gets a handle to the TextEdit structure, gets the view rectangle, and assigns a value to the h field of a point variable equal to the left of the view rectangle plus 4 pixels.

The switch executes only if the mouse-down is not in the scroll box.

In the case of the Up scroll arrow, the variable delta is assigned a value which will ensure that, after the scroll, the top of the incoming line of text will be positioned cleanly at top of the view rectangle.

In the case of the Down scroll arrow, the variable delta is assigned a value which will ensure that, after the scroll, the bottom of the incoming line of text will be positioned cleanly at bottom of the view rectangle.

In the case of the Up gray area/track, the variable delta is assigned a value which will ensure that, after the scroll, the top of the top line of text will be positioned cleanly at the top of the view rectangle and the line of text which was previously at the top will still be visible at the bottom of the view rectangle.

In the case of the Down gray area/track, the variable delta is assigned a value which will ensure that, after the scroll, the bottom of the bottom line of text will be positioned cleanly at the bottom of the view rectangle and the line of text which was previously at the bottom will still be visible at the top of the view rectangle.

The first line after the switch gets the pre-scroll scroll bar value. If the text is not fully scrolled up and a scroll up is called for, or if the text is not fully scrolled down and a scroll down is called for, the current clipping region is saved, the clipping region is set to the dialog's port rectangle, the scroll bar value is set to the required new value, and the saved clipping region is restored. (TextEdit may have set the clipping region to the view rectangle, so it must be changed to include the scroll bar area, otherwise the scroll bar will not be drawn.)

With the scroll bar's new value set and the scroll box redrawn in its new position, the function for scrolling the text and pictures is called. Note that this last line will also be called if the mouse-down was within the scroll box.

doScrollTextAndPicts

doScrollTextAndPicts is called from actionFunction. It scrolls the text within the view rectangle and calls another function to draw any picture whose rectangle intersects the "vacated" area of the view rectangle.

The first line sets the background colour to white and the background pattern to white.

The next two lines get a handle to the TextEdit structure. The next line determines the difference between the top of the destination rectangle and the top of the view rectangle and the next subtracts from this value the scroll bar's new value. If the result is zero, the text must be fully scrolled in one direction or the other, so the function simply returns.

If the text is not already fully scrolled one way or the other, TEScroll scrolls the text in the view rectangle by the number of pixels determined at the fifth line.

If there are no pictures associated with the 'TEXT' resource in use, the function returns immediately after the text is scrolled.

The next consideration is the pictures and whether any of their rectangles, as stored in the picture information structure, intersect the area of the view rectangle "vacated" by the scroll. At the if/else block, a rectangle is made equal to the "vacated" area of the view rectangle, the if block catering for the scrolling up case and the else block catering for the scrolling down case. This rectangle is passed as a parameter in the call to drawPictures.

doDrawPictures

doDrawPictures determines whether any pictures intersect the rectangle passed to it as a formal parameter and draws any pictures that do.

The first two lines get handles to the dialog's document structure and the TextEdit structure.

The next line determines the difference between the top of the destination rectangle and the top of the view rectangle. This will be used later to offset the picture's rectangle from destination rectangle coordinates to view rectangle coordinates. The next line determines the number of pictures associated with the current 'TEXT' resource, a value which will be used to control the number of passes through the following for loop.

Within the loop, the picture's rectangle is retrieved from the picture information structure and offset to the coordinates of the view rectangle. SectRect determines whether this rectangle intersects the rectangle passed to drawPictures from scrollTextAndPicts. If it does not, the loop returns for the next iteration. If it does, the picture's handle is retrieved, LoadResource checks whether the 'PICT' resource is in memory and, if necessary, loads it, HLock locks the handle, DrawPicture draws the picture, and HUnlock unlocks the handle. Before DrawPicture is called, the clipping region is temporarily adjusted to equate to the rectangle passed to drawPictures from scrollTextAndPicts so as to limit drawing to that rectangle.

doCloseHelp

doCloseHelp closes down the Help dialog.

The first two lines retrieve a handle to the dialog's document structure. The next two lines dispose of the region used to save the clipping region. TEDispose disposes of the TextEdit structure. The next block disposes of any 'PICT' resources currently in memory, together with the picture information structure. Finally, the window's document structure is disposed of, the window itself is disposed of, and the graphics port saved in doHelp is set as the current port.

doSetBackgroundWhite

doSetBackgroundWhite sets the background colour to white and the background pattern to the pattern white.

Demonstration Program DateTimeNumbers Listing

```
// *******************************************************************************************
// DateTimeNumbers.c                                                       CARBON EVENT MODEL
// *******************************************************************************************
//
// This program, which opens a single modeless dialog, demonstrates the formatting and display
// of dates, times and numbers.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple/Application, File, and Edit menus
//    (preload,  non-purgeable).
//
// •  A 'DLOG' resource and associated 'dlgx', 'DITL', 'dfnt', and 'CNTL' resources
//    (purgeable).
//
// •  'hdlg' and 'STR#' resources (purgeable) for balloon help and help tags.
//
// •  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 mEdit                   130
#define  iCut                   3
#define  iCopy                  4
#define  iPaste                 5
#define  iClear                 6
#define rDialog                 128
#define  iStaticTextTodaysDate  2
#define  iStaticTextCurrentTime 4
#define  iEditTextTitle         10
#define  iEditTextQuantity      11
#define  iEditTextValue         12
#define  iEditTextDate          13
#define  iButtonEnter           18
#define  iButtonClear           19
#define  iStaticTextTitle       26
#define  iStaticTextQuantity    27
#define  iStaticTextUnitValue   28
#define  iStaticTextTotalValue  29
#define  iStaticTextDate        30
#define kReturn                 0x0D
#define kEnter                  0x03
#define kLeftArrow              0x1C
#define kRightArrow             0x1D
#define kUpArrow                0x1E
#define kDownArrow              0x1F
#define kBackspace              0x08
#define kDelete                 0x7F
#define topLeft(r)              (((Point *) &(r))[0])
#define botRight(r)             (((Point *) &(r))[1])
```

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

Boolean         gRunningOnX = false;
DialogRef       gDialogRef;
DateCacheRecord gDateCacheRec;
Boolean         gInBackground;

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

void                    main                (void);
void                    doPreliminaries     (void);
OSStatus                appEventHandler     (EventHandlerCallRef,EventRef,void *);
OSStatus                windowEventHandler  (EventHandlerCallRef,EventRef,void *);
void                    doIdle              (void);
void                    doMenuChoice        (MenuID,MenuItemIndex);
void                    doCopyPString       (Str255,Str255);
void                    doTodaysDate        (void);
void                    doAcceptNewRecord   (void);
void                    doUnitAndTotalValue (Str255,Str255);
void                    doDate              (Str255);
void                    doAdjustCursor      (WindowRef);
void                    doClearAllFields    (void);
ControlKeyFilterResult  numericFilter       (ControlRef,SInt16 *,SInt16 *,EventModifiers *);
void                    helpTags            (void);

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

void  main(void)
{
  MenuBarHandle       menubarHdl;
  SInt32              response;
  MenuRef             menuRef;
  ControlKeyFilterUPP numericFilterUPP;
  ControlRef          controlRef;
  EventTypeSpec   applicationEvents[] = { { kEventClassApplication, kEventAppActivated    },
                                          { kEventClassCommand,     kEventProcessCommand  },
                                          { kEventClassMouse,       kEventMouseMoved      } };
  EventTypeSpec   windowEvents[]      = { { kEventClassWindow,   kEventWindowDrawContent  },
                                          { kEventClassWindow,   kEventWindowActivated    },
                                          { kEventClassWindow,   kEventWindowClose        },
                                          { kEventClassMouse,    kEventMouseDown          },
                                          { kEventClassKeyboard, kEventRawKeyDown         } };

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

  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);
    }

    gRunningOnX = true;
  }
  else
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
      SetMenuItemCommandID(menuRef,iQuit,kHICommandQuit);
  }

   // ............................................................ install application event handler and a timer

  InstallApplicationEventHandler(NewEventHandlerUPP((EventHandlerProcPtr) appEventHandler),
                                 GetEventTypeCount(applicationEvents),applicationEvents,
                                 0,NULL);

  InstallEventLoopTimer(GetCurrentEventLoop(),0,TicksToEventTime(GetCaretTime()),
                        NewEventLoopTimerUPP((EventLoopTimerProcPtr) doIdle),NULL,NULL);

  // ........................................ open modeless dialog, change attributes, and install handler

  if(!(gDialogRef = GetNewDialog(rDialog,NULL,(WindowRef) -1)))
    ExitToShell();

  ChangeWindowAttributes(GetDialogWindow(gDialogRef),kWindowStandardHandlerAttribute |
                                                     kWindowCloseBoxAttribute,
                                                     kWindowCollapseBoxAttribute);

  InstallWindowEventHandler(GetDialogWindow(gDialogRef),
                            NewEventHandlerUPP((EventHandlerProcPtr) windowEventHandler),
                            GetEventTypeCount(windowEvents),windowEvents,0,NULL);

  // ......... create universal procedure pointers for key filter, attach to two edit text controls

  numericFilterUPP = NewControlKeyFilterUPP((ControlKeyFilterProcPtr) numericFilter);

  GetDialogItemAsControl(gDialogRef,iEditTextQuantity,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlEditTextKeyFilterTag,
                 sizeof(numericFilterUPP),&numericFilterUPP);

  GetDialogItemAsControl(gDialogRef,iEditTextValue,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlEditTextKeyFilterTag,
                 sizeof(numericFilterUPP),&numericFilterUPP);

  // ........................................ set help tags, get and display today's date and show window

  if(gRunningOnX)
    helpTags();

  doTodaysDate();

  ShowWindow(GetDialogWindow(gDialogRef));

  // ............................................ display today's date and initialise date cache structure

  InitDateCache(&gDateCacheRec);

  // ................................................................................ run application event loop

  RunApplicationEventLoop();
}

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

void  doPreliminaries(void)
{
  MoreMasterPointers(64);
  InitCursor();
}

// *************************************************************************** appEventHandler

OSStatus  appEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                          void * userData)
{
  OSStatus      result = eventNotHandledErr;
  UInt32        eventClass;
  UInt32        eventKind;
  HICommand     hiCommand;
  MenuID        menuID;
  MenuItemIndex menuItem;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  switch(eventClass)
  {
    case kEventClassApplication:
      if(eventKind == kEventAppActivated)
        SetThemeCursor(kThemeArrowCursor);
      break;

    case kEventClassCommand:
      if(eventKind == kEventProcessCommand)
      {
        GetEventParameter(eventRef,kEventParamDirectObject,typeHICommand,NULL,
                          sizeof(HICommand),NULL,&hiCommand);
        menuID = GetMenuID(hiCommand.menu.menuRef);
        menuItem = hiCommand.menu.menuItemIndex;
        if((hiCommand.commandID != kHICommandQuit) &&
           (menuID >= mAppleApplication && menuID <= mEdit))
        {
          doMenuChoice(menuID,menuItem);
          result = noErr;
        }
      }
      break;

    case kEventClassMouse:
      if(eventKind == kEventMouseMoved)
      {
        doAdjustCursor(GetDialogWindow(gDialogRef));
        result = noErr;
      }
      break;
  }

  return result;
}

// ********************************************************************** windowEventHandler

OSStatus  windowEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                             void* userData)
{
  OSStatus    result = eventNotHandledErr;
  UInt32      eventClass;
  UInt32      eventKind;
  EventRecord eventRecord;
  SInt16      itemHit;
  SInt8       charCode;
  ControlRef  controlRef;
  UInt32      finalTicks;

  eventClass = GetEventClass(eventRef);
  eventKind  = GetEventKind(eventRef);

  switch(eventClass)
  {
    case kEventClassWindow:                                             // event class window
      ConvertEventRefToEventRecord(eventRef,&eventRecord);
      switch(eventKind)
      {
        case kEventWindowActivated:
          DialogSelect(&eventRecord,&gDialogRef,&itemHit);
          result = noErr;
          break;

        case kEventWindowClose:
          QuitApplicationEventLoop();
          result = noErr;
          break;
      }

    case kEventClassMouse:                                              // event class mouse
      ConvertEventRefToEventRecord(eventRef,&eventRecord);
      switch(eventKind)
      {
        case kEventMouseDown:
          if(IsDialogEvent(&eventRecord))
          {
            if(DialogSelect(&eventRecord,&gDialogRef,&itemHit))
            {
              if(itemHit == iButtonEnter)
              {
                doAcceptNewRecord();
                doClearAllFields();
              }
              else if(itemHit == iButtonClear)
                doClearAllFields();
            }
          }
          doAdjustCursor(GetDialogWindow(gDialogRef));
          break;
      }
      break;

    case kEventClassKeyboard:                                        // event class keyboard
      switch(eventKind)
      {
        case kEventRawKeyDown:
          ConvertEventRefToEventRecord(eventRef,&eventRecord);
          GetEventParameter(eventRef,kEventParamKeyMacCharCodes,typeChar,NULL,
                            sizeof(charCode),NULL,&charCode);
          if((charCode == kReturn) || (charCode == kEnter))
          {
            GetDialogItemAsControl(gDialogRef,iButtonEnter,&controlRef);
            HiliteControl(controlRef,kControlButtonPart);
            Delay(8,&finalTicks);
            HiliteControl(controlRef,kControlEntireControl);
            doAcceptNewRecord();
            doClearAllFields();
            return noErr;
          }
          break;
      }
      break;
  }

  return result;
}

// ********************************************************************************** doIdle

void  doIdle(void)
{
  UInt32        rawSeconds;
  static UInt32 oldRawSeconds;
  Str255        timeString;
  ControlRef    controlRef;
  if(gRunningOnX)
    IdleControls(GetDialogWindow(gDialogRef));

  GetDateTime(&rawSeconds);

  if(rawSeconds > oldRawSeconds)
  {
    TimeString(rawSeconds,true,timeString,NULL);

    GetDialogItemAsControl(gDialogRef,iStaticTextCurrentTime,&controlRef);
    SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,timeString[0],
                   &timeString[1]);
    Draw1Control(controlRef);

    oldRawSeconds = rawSeconds;
  }
}

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

void  doMenuChoice(MenuID menuID,MenuItemIndex menuItem)
{
  if(menuID == 0)
    return;

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

    case mEdit:
      switch(menuItem)
      {
        case iCut:
          DialogCut(gDialogRef);
          break;

        case iCopy:
          DialogCopy(gDialogRef);
          break;

        case iPaste:
          DialogPaste(gDialogRef);
          break;

        case iClear:
          DialogDelete(gDialogRef);
          break;
      }
      break;
  }
}

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

void  doCopyPString(Str255 sourceString,Str255 destinationString)
{
  SInt16 stringLength;

  stringLength = sourceString[0];
  BlockMove(sourceString + 1,destinationString + 1,stringLength);
  destinationString[0] = stringLength;
}

// ******************************************************************************** doTodaysDate

void  doTodaysDate(void)
```

```
{
  UInt32     rawSeconds;
  Str255     dateString;
  ControlRef controlRef;

  GetDateTime(&rawSeconds);
  DateString(rawSeconds,longDate,dateString,NULL);

  GetDialogItemAsControl(gDialogRef,iStaticTextTodaysDate,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,dateString[0],
                 &dateString[1]);
}

// ********************************************************************** doAcceptNewRecord

void  doAcceptNewRecord(void)
{
  SInt16     theType;
  Handle     theHandle;
  Rect       theRect;
  Str255     titleString, quantityString, valueString, dateString;
  ControlRef controlRef;

  GetDialogItem(gDialogRef,iEditTextTitle,&theType,&theHandle,&theRect);
  GetDialogItemText(theHandle,titleString);

  GetDialogItem(gDialogRef,iEditTextQuantity,&theType,&theHandle,&theRect);
  GetDialogItemText(theHandle,quantityString);

  GetDialogItem(gDialogRef,iEditTextValue,&theType,&theHandle,&theRect);
  GetDialogItemText(theHandle,valueString);

  GetDialogItem(gDialogRef,iEditTextDate,&theType,&theHandle,&theRect);
  GetDialogItemText(theHandle,dateString);

  if(titleString[0] == 0 || quantityString[0] == 0 || valueString[0] == 0 ||
     dateString[0] == 0)
  {
    SysBeep(10);
    return;
  }

  GetDialogItemAsControl(gDialogRef,iStaticTextTitle,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,titleString[0],
                 &titleString[1]);
  Draw1Control(controlRef);

  GetDialogItemAsControl(gDialogRef,iStaticTextQuantity,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,quantityString[0],
                 &quantityString[1]);
  Draw1Control(controlRef);

  doUnitAndTotalValue(valueString,quantityString);

  doDate(dateString);
}

// ******************************************************************** doUnitAndTotalValue

void  doUnitAndTotalValue(Str255 valueString, Str255 quantityString)
{
  Handle           itl4ResourceHdl;
  SInt32           numpartsOffset;
  SInt32           numpartsLength;
  NumberParts      *numpartsTablePtr;
  Str255           formatString = "\p'$'###,###,###.00;'Valueless';'Valueless'";
  NumFormatString  formatStringRec;
  Str255           formattedNumString;
  extended80       value80Bit;
  SInt32           quantity;
  double           valueDouble;
  FormatResultType result;
  ControlRef       controlRef;

  GetIntlResourceTable(smSystemScript,iuNumberPartsTable,&itl4ResourceHdl,&numpartsOffset,
                       &numpartsLength);
  numpartsTablePtr = (NumberPartsPtr) ((SInt32) *itl4ResourceHdl + numpartsOffset);

  StringToFormatRec(formatString,numpartsTablePtr,&formatStringRec);

  StringToExtended(valueString,&formatStringRec,numpartsTablePtr,&value80Bit);
  ExtendedToString(&value80Bit,&formatStringRec,numpartsTablePtr,formattedNumString);

  GetDialogItemAsControl(gDialogRef,iStaticTextUnitValue,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,
                 formattedNumString[0],&formattedNumString[1]);
  Draw1Control(controlRef);

  StringToNum(quantityString,&quantity);

  valueDouble = x80tod(&value80Bit);
  valueDouble = valueDouble * quantity;
  dtox80(&valueDouble,&value80Bit);

  result = ExtendedToString(&value80Bit,&formatStringRec,numpartsTablePtr,
                            formattedNumString);

  if(result == fFormatOverflow)
    doCopyPString("\p(Too large to display)",formattedNumString);

  GetDialogItemAsControl(gDialogRef,iStaticTextTotalValue,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,
                 formattedNumString[0],&formattedNumString[1]);
  Draw1Control(controlRef);
}

// ************************************************************************************ doDate

void  doDate(Str255 dateString)
{
  SInt32       lengthUsed;
  LongDateRec  longDateTimeRec;
  LongDateTime longDateTimeValue;
  ControlRef   controlRef;

  longDateTimeRec.ld.hour = 0;
  longDateTimeRec.ld.minute = 0;
  longDateTimeRec.ld.second = 0;

  StringToDate((Ptr) dateString + 1,dateString[0],&gDateCacheRec,&lengthUsed,
               &longDateTimeRec);

  LongDateToSeconds(&longDateTimeRec,&longDateTimeValue);
  LongDateString(&longDateTimeValue,longDate,dateString,NULL);

  GetDialogItemAsControl(gDialogRef,iStaticTextDate,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlStaticTextTextTag,dateString[0],
                 &dateString[1]);
  Draw1Control(controlRef);
}

// **************************************************************************** doAdjustCursor

void  doAdjustCursor(WindowRef windowRef)
{
  GrafPtr    oldPort;
  RgnHandle  arrowRegion,iBeamRegion;
  ControlRef controlRef;
  Rect       iBeamRect;
  Point      mouseLocation;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  arrowRegion = NewRgn();
  iBeamRegion = NewRgn();

  SetRectRgn(arrowRegion,-32768,-32768,32767,32767);

  GetKeyboardFocus(windowRef,&controlRef);
  GetControlBounds(controlRef,&iBeamRect);

  LocalToGlobal(&topLeft(iBeamRect));
  LocalToGlobal(&botRight(iBeamRect));

  RectRgn(iBeamRegion,&iBeamRect);
  DiffRgn(arrowRegion,iBeamRegion,arrowRegion);

  GetMouse(&mouseLocation);
  LocalToGlobal(&mouseLocation);

  if(PtInRgn(mouseLocation,iBeamRegion))
    SetThemeCursor(kThemeIBeamCursor);
  else
    SetThemeCursor(kThemeArrowCursor);

  DisposeRgn(arrowRegion);
  DisposeRgn(iBeamRegion);

  SetPort(oldPort);
}

// ************************************************************************** doClearAllFields

void  doClearAllFields(void)
{
  SInt16     a;
  ControlRef controlRef;
  Str255     theString = "\p";

  for(a = iEditTextTitle;a <= iEditTextDate;a++)
  {
    GetDialogItemAsControl(gDialogRef,a,&controlRef);
    SetControlData(controlRef,kControlEntireControl,kControlEditTextTextTag,theString[0],
                   &theString[1]);
    Draw1Control(controlRef);

    if(a == iEditTextTitle)
      SetKeyboardFocus(GetDialogWindow(gDialogRef),controlRef,kControlFocusNextPart);
  }
}

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

ControlKeyFilterResult  numericFilter(ControlRef controlRef,SInt16* keyCode,SInt16 *charCode,
                                      EventModifiers *modifiers)
{
  if(((char) *charCode >= '0') && ((char) *charCode <= '9') || (char) *charCode == '.' ||
     (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;
}

// ******************************************************************************** helpTags

void  helpTags(void)
{
  HMHelpContentRec helpContent;
  SInt16           a;
  static SInt16    itemNumber[7] = { 1,3,21,22,23,24,25 };
  ControlRef       controlRef;

  HMSetTagDelay(5);
  HMSetHelpTagsDisplayed(true);
  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideTopCenterAligned;

  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 128;

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

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

Demonstration Program DateTimeNumbers Comments

When this program is run, the user should enter data in the four edit text controls, using the tab key or mouse clicks to select the required control and pressing the Return key or clicking the Enter Record button when data has been entered in all controls. Note that numeric filters are used in the Quantity and Value edit text controls.

In order to observe number formatting effects, the user should occasionally enter very large numbers and negative numbers in the Value field. In order to observe the effects of date string parsing and formatting, the user should enter dates in a variety of formats, for example: "2 Mar 95", "2/3/95", "March 2 1995", "2 3 95", etc.

Global Variables

gDateCacheRec is used within the function doDate.

main

The call to InstallEventLoopTimer installs a timer which will fire repeatedly at the interval returned by the call to GetCaretTime. When the timer fires, the function doIdle is called. In addition to calling IdleControls, doIdle updates the current time displayed in a static text control in the top of the dialog.

doTadaysDate is called to get the date and set it in a static text control at the top of the dialog.

In the function doDate, the function that creates the long date-time structure takes an initialised date cache structure as a parameter. The call to InitDateCache initialises a date cache structure.

windowEventHandler

Note that all events are passed to DialogSelect.

When the kEventMouseDown event is received, if the Enter Record push button was hit, the function doAcceptNewRecord is called, following which doClearAllFields is called to clear all of the edit text controls. The same occurs when the kEventRawKeyDown event is received if the key pressed was Return or Enter.

doIdle

doIdle, which is called when the timer fires, blinks the insertion point caret and sets the current time in the static text control at top-right in the dialog.

If the program is running on Mac OS 8/9, IdleControls is called to ensure that the caret blinks regularly in the edit text control with current keyboard focus. (On Mac OS X, these controls have their own in-built timers.)

GetDateTime retrieves the "raw" seconds value, as known to the system. (This is the number of seconds since 1 Jan 1904.) If that value is greater than the value retrieved the last time doIdle was called, TimeString converts the raw seconds value to a string containing the time formatted according to flags in the numeric format ('itl0') resource. (Since NULL is specified in the resource handle parameter, the appropriate 'itl0' resource for the current script system is used.) This string is then set in the static text control, following which Draw1Control is called to redraw the control. The retrieved raw seconds value is assigned to the static variable oldRawSeconds for use next time doIdle is called.

doTodaysDate

doTodaysDate sets the date in the static text control at top-left of the dialog.

GetDateTime gets the raw seconds value, as known to the system. DateString converts the raw seconds value to a string containing a date formatted in long date format according to flags in the numeric format ('itl0') resource. (Since NULL is specified in the resource handle parameter, the appropriate 'itl0' resource for the current script system is used.) This string is then set in the static text control.

doAcceptNewRecord

doAcceptNewRecord is called when the Return or Enter key is pressed, or when the Enter Record button is clicked. Assuming each edit text control contains at least one character of text, it calls other functions to format (where necessary) and display strings in the "Last Record Entered" group box area.

The calls to GetDialogItem get the handle in the hText field of each edit text control's TextEdit structure, allowing the calls to GetDialogItemText to get the text into four local variables of type Str255.

If the length of any of these strings is 0, the system alert sound is played and doAcceptNewRecord returns.

The text from the Item Title and Quantity edit text controls are set in the relevant static text controls within the Last Record Entered group box, and DrawIControl is called to draw those controls. doUnitAndTotalValue and doDate are then called.

doUnitAndTotalValue

doUnitAndTotalValue is called by doAcceptNewRecord to convert the string from the Value edit text control to a floating point number, convert that number to a formatted number string, set that string in the relevant static text control, convert the string from the Quantity edit text control to an integer, multiply the floating point number by the integer to arrive at the "Total Value" value, convert the result to a formatted number string, and set that string in the relevant static text control.

A pointer to a number parts table is required by the functions that convert between floating point numbers and strings. Accordingly, the first three lines get the required pointer.

StringToFormatRec converts the number format specification string into the internal numeric representation required by the functions that convert between floating point numbers and strings.

StringToExtended converts the received Value string into a floating point number of type extended (80 bits). ExtendedToString converts that number back to a string, formatted according to the internal numeric representation of the number format specification string. That string is then set in the relevant static text control and Draw1Control is called to draw that control.

The intention now is to multiply the quantity by the unit value to arrive at a total value. The string received in the quantityString formal parameter is converted to an integer value of type SInt32 by StringToNum. The extended80 value is converted to a value of type double before the multiplication occurs. The result of the multiplication is assigned to the variable of type double. This is then converted to an extended80.

The extended80 value is then passed in the first parameter of ExtendedToString for conversion to a formatted string. If ExtendedToString does not return fFormatOverflow, the formatted string is set in the relevant static text control and Draw1Control is called to draw that control.

doDate

doDate is called by doAcceptNewRecord to create a long date-time structure from the string in the "Date" edit text control, format the date as a string (long date format), and set that string in the relevant static text control.

A pointer to the string containing the date as entered by the user, and the length of that string, are passed in the call to StringToDate. StringToDate parses the input string and fills in the relevant fields of the long date-time structure.

The function StringToDate fills in only the year, month, day, and day of the week fields of a long date-time structure. The function StringToTime fills in the hour, minute, and second. If you do not call StringToTime, as is the case here, you need to zero the time-related fields of the long date-time structure. If this is not done, the call to LongDateToSeconds will return an erroneous value. (LongDateToSeconds always assumes that all the fields of the long date-time structure passed to it are valid.)

LongDateToSeconds converts the long date-time structure to a long date-time value. The long date-time value is then passed as a parameter to LongDateString, which converts the long date-time value to a long format date string formatted according to the specified international resource. (In this case, NULL is passed as the international resource parameter, meaning that the appropriate 'itl1' resource for the current script system is used.)

The formatted date string is then set in the relevant static text control and Draw1Control is called to draw that control.