19

TEXT AND TEXTEDIT

Includes Demonstration Programs Text1 and Text2

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, 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 — Drawing With QuickDraw, which included descriptions of QuickDraw functions used for drawing text and for setting the font, style, size, and transfer mode. In addition, Chapter 15 — Printing contained a brief treatment of considerations applying to the printing of text.

This chapter addresses:

- TextEdit, which is a collection of functions and data structures you can use to provide your application with basic text editing and formatting capabilities.
- 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.

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**.

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

The **Apple Standard Roman character set** is the fundamental character set for the Macintosh computer. It uses all character codes from 0x00 to 0xFF, and includes the uppercase versions of all of the lowercase accented Roman characters, a number of symbols, and other forms (see Fig 1). The Standard Roman character set is 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	¶	∂	Δ	÷	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

You never see a character on a display device. What you see on a display device is a **glyph**, which is the visual representation of a character. In other words, a glyph is the exact shape by which a character is represented. A specific character can be represented by many different shapes (that is, glyphs).

The Font Manager uses two types of glyphs: **bitmapped glyphs** and glyphs from **outline fonts**. A bitmapped glyph is a bitmap designed at a fixed size for a particular display device. A glyph from an outline font is a model of how the glyph should look. The "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 particular character set share certain design characteristics, they form a **typeface**, which is a distinctively designed collection of glyphs. Each typeface has its own name, such as New York, Geneva, or Times. The same typeface can be used with different hardware, such as typesetting machines, monitors, and laser printers.

Styles

A **style** is a specific variation in the appearance of a glyph which can be applied consistently to all glyphs in a typeface. Styles available on the Macintosh include plain, bold, italic, underline, outline, shadow, condensed, and extended. QuickDraw can add styles to bitmaps, or a font designer can design a font in a specific style (for example, Courier Bold).

Fonts and Font Families

A **font** refers to a complete set of glyphs in a specific typeface and style — and, in the case of bitmapped fonts, a specific size. All fonts have a font name, which is stored in a string such as "Geneva" or "New York". The font name is usually the same name as the typeface from which it was derived. If a font is not in the plain style, its style becomes part of the font name, for example "Palatino Bold".

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

- Bitmapped fonts are fonts of the 'FONT' resource type (the original resource type for fonts) and the bitmapped font ('NFNT') resource type. These resources provide a separate bitmap for each glyph in each size and style.

- Outline fonts are fonts of the outline font ('sfnt') resource type which consist of glyphs in a particular typeface and style with no size restriction. The outline font resource type emerged at the time of the addition of TrueType outline font support with System 7.

When multiple fonts of the same typeface are present in the system software, the Font Manager groups them into **font families** of the font family ('FOND') resource type. A **font family ID** is the resource ID for a font family. Because there are so many font families available for the Macintosh, many families have the same ID.[2]

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.[3] 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

Monospaced and Proportional Fonts. 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** is an imaginary horizontal line chosen by the font's designer which corresponds approximately with the tops of the uppercase letters of the font. The **descent line** is an imaginary line which usually corresponds to the bottom of the descenders (the tails of glyphs like "p" and "g").

Glyph Origin and Advance Width. QuickDraw begins drawing a glyph at the **glyph origin**. There is some white space between the glyph origin and the beginning of the glyph called the **left side bearing**. The **advance width** is the full horizontal measurement of a glyph as measured from its glyph origin to the glyph origin of the next glyph.

2 This is the reason why your application should refer to fonts by name and not by number when it stores font references in a document.

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

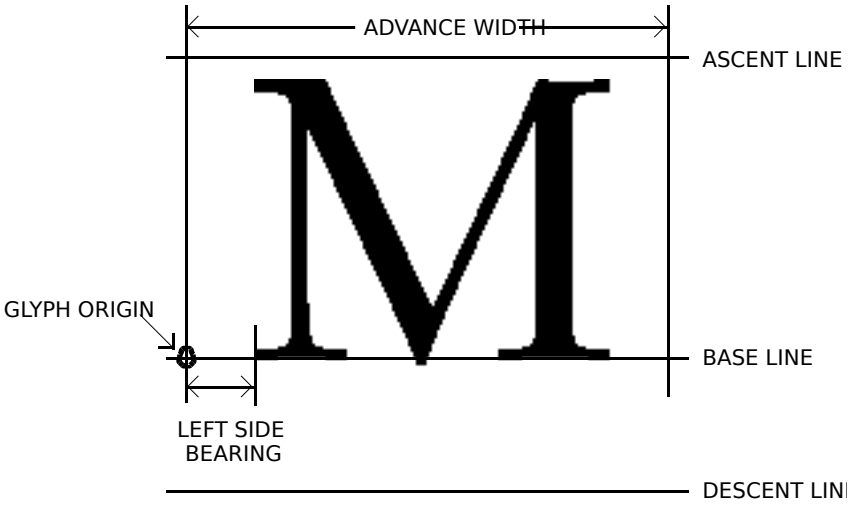

FIG 2 - FONT MEASUREMENT TERMS

Font Size. Font size indicates the size of the font's glyphs as measured from the base line of one line of the text to the base line of the next line. Font size is measured in **points** (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 leading value is the amount of blank vertical space between the descent line of one line and the ascent line of the next line.)

The base line, ascent line, descent line, and glyph origin are illustrated at Fig 2.

System Font and Application Font

Macintosh system software recognises the following two special fonts, which should always be present:

- The **system font**, which is used for menus, dialog boxes, and other messages to the user from the Finder and the Operating System. The system font is either 12 point Charcoal or 12-point Chicago, depending on the user's setting in the Appearance control panel.

- The **application font**, which is the suggested default font for use by monostyled TextEdit and by applications which do not support user selection of fonts. The application font is 12-point Geneva.

The system font and application font have **special font designators**. The system font designator is 0 and the application font designator is 1. These special designators are *not* actual font family resource ID numbers and cannot be used as such in Resource Manager calls; however, they can be used in place of the font family ID in the `txFont` field of the colour graphics port and in text-related calls that take a font family ID. The system maps the special designators to the actual font family IDs.

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 bar which marks the insertion point of text and the **cursor** is the arrow, I-beam or other icon that moves with the mouse.

A caret position is a location on the screen which 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, it displaces the character at the insertion point, shifting it and all subsequent characters in the buffer forward by one position.

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

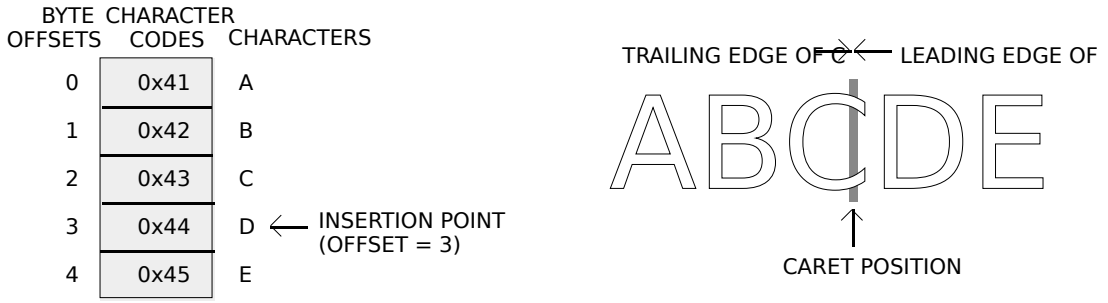

FIG 3 - CARET POSITION AND INSERTION POINT

Converting Screen Position to Text Offset

A mouse-down event can occur anywhere within the area of a glyph, but the caret position which is derived from that event must be an infinitesimally thin line falling between the two glyphs.

As shown at Fig 4, a line of displayed glyphs is divided into a series of **mouse-down regions**. A mouse-down region is a screen area within which any mouse click will yield the same caret position. It extends from the centre of one glyph to the centre of the next glyph (except at the ends of lines).

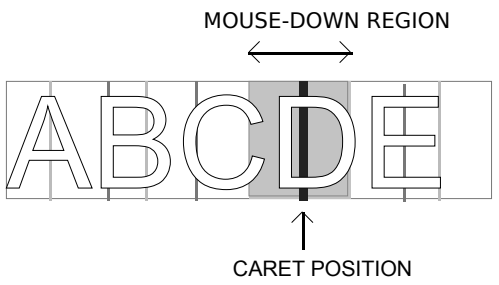

FIG 4 - INTERPRETING CARET POSITION FROM A M

Selection Range and Insertion Points

The **selection range** is the sequence of zero or more characters, contiguous in memory, where the next editing operation is to occur. A selection range of zero characters 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 B, C, and D at Fig 3 were highlighted, the highlighting would extend from the leading edge of B (offset = 1) to the leading edge of E (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 grey, unblinking caret is displayed. By default, outline highlighting is disabled.

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 give your application basic text formatting and editing capabilities. Its functions can be used in applications such as spreadsheets, on-line (data entry) forms, simple text editors, and drawing and painting programs with simple text-editing features. TextEdit relies on Script Manager, QuickDraw, and Text Utilities functions to handle text correctly, eliminating the need for your application to call these functions directly.

TextEdit was originally designed to handle editable text items in dialog boxes and other parts of the system software. It was subsequently enhanced to support some of the cumbersome tasks that a text processor needs to perform. That said, it 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.

Editing Tasks Performed by TextEdit

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

- Selection of text by clicking and dragging the mouse.
- Double-clicking to select words.
- Extending or shortening selection ranges by Shift-clicking.
- Highlighting the current text selection, or displaying a blinking vertical bar (the caret) at the insertion point.
- Line breaking, that is, preventing a word from being split inappropriately between lines when text is drawn.
- Cutting, copying, and pasting within and between applications.
- Managing the use of more than one font, size, colour and stylistic variation from character to character.

Thus, if you do not need to manipulate large files and do not need extensive formatting capabilities, TextEdit is a convenient alternative to writing your own specialised text processing functions.

TextEdit Options

You can use TextEdit at different levels of complexity.

Using TextEdit Indirectly

For the simplest level of text handling (that is, in dialog boxes), 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 edit structure (see below) because TETextBox creates its own edit 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 the application font (if you do not allow the user to select the font) or with any single available font (if you do allow user selection).

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 must use **multistyled TextEdit**.

Caret Position and Movement in TextEdit

TextEdit marks the position in the displayed text where the next editing operation will occur with the caret. When TextEdit pastes text into a structure, it positions the caret after the newly pasted text. When the user presses the Up Arrow key or the Down Arrow key, the caret moves up or down one line respectively. When the caret is on the first line of an edit structure, and the user presses the Up Arrow key, TextEdit moves the caret to the beginning of text on that line. When the caret is on the last line of an edit structure, and the user presses the Down Arrow key, TextEdit moves the caret to the end of the text on that line.[4]

If spaces at the end of a text line extend beyond the view rectangle (see below), TextEdit draws the caret at the edge of the view rectangle, not beyond it. Whether TextEdit displays a caret at the beginning or end of a line when a mouse-down event occurs at a

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

line's end depends on the current caret position and the value in the clikStuff field of the edit structure. TextEdit sets this field to reflect whether the most recent mouse-down event occurred on the leading or trailing edge of a glyph. For example, if the mouse-down event occurs on the leading edge of a glyph, TextEdit displays the caret at the caret position corresponding to the leading edge of the glyph. If the mouse-down event is on the trailing edge of a glyph, TextEdit displays the caret at the beginning of the next line.

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. 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 position in an application's scrollbars to follow up the scrolling. The default click loop function can, however, be replaced with an application-defined click loop function which accommodates scroll bars.

TextEdit 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.

Private Scrap

The private scrap, which belongs to your application, is used for all cut, copy, and paste activity. When the text is multistyled, TextEdit also copies the text to the Scrap Manager's desk scrap.

Null Scrap

The null scrap is used to store **character attribute** information[5] associated with a null selection (that is, an insertion point) or text that is deleted when the user backspaces over it.

Character attribute information is retained in the null scrap until it is used, that is, when it is applied to newly inserted text, or until some other editing action renders it unnecessary, such as when TextEdit sets a new selection range. A number of functions which deal with multistyled text check the null scrap for character attribute information and, if there is any, apply it to the newly inserted text when character attributes for that text are not available.

TextEdit creates and initialises a null scrap for a multistyled edit structure when an application creates the edit structure. The null scrap remains throughout the life of the edit structure, being disposed of when the application disposes of the edit structure and release the memory associated with it.

Style Scrap

When you cut or copy multistyled text, memory is allocated dynamically for the style scrap and the character attribute information is copied to it. Your application can also use the style scrap as follows:

- To save and restore multistyled text, both the text and the associated character attribute information must be preserved. You can save character attributes associated with a range of text in the style scrap.

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

- You can create a style scrap structure and store character attribute information in it to be applied to inserted text.

Text Alignment

The term **alignment** means the horizontal alignment of lines of text with respect to the left and right edges of the text area. Alignment can be left-aligned, right-aligned, centred, or justified (that is, aligned with both the left and right edges of the text area). Justification is achieved by spreading or compressing text to fit a given line width. TextEdit supports left-aligned, right-aligned and centred alignments.

Customising TextEdit

TextEdit may be customised by replacing the end-of-line, drawing, width-measuring, and hit test default hook functions using TECustomHook. You can also customise word selection, automatic scrolling, and how to determine the length of a line in order to justify it.

Primary TextEdit Data Structures

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

The Edit Structure

The edit structure is the principal data structure used by TextEdit. The structure of the edit structure is the same regardless of whether the text is monostyled or multistyled, although some fields are used differently for multistyled edit structures. The edit 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 edit struct.
    short           fontAscent;     // Font ascent. -1 in multistyled edit 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 edit structure.
    GrafPtr         inPort;         // Pointer to grafPort for this edit 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 — Destination rectangle, in local coordinates.

viewRect — View rectangle, in local coordinates.

When you allocate an edit structure, you specify where the text is to be drawn and where it is to be made visible. The **destination rectangle** is the area in which text is drawn and the **view rectangle** is that portion of the window within which the text is actually displayed. Fig 5 illustrates the relationship between the destination rectangle and the view rectangle.[6]

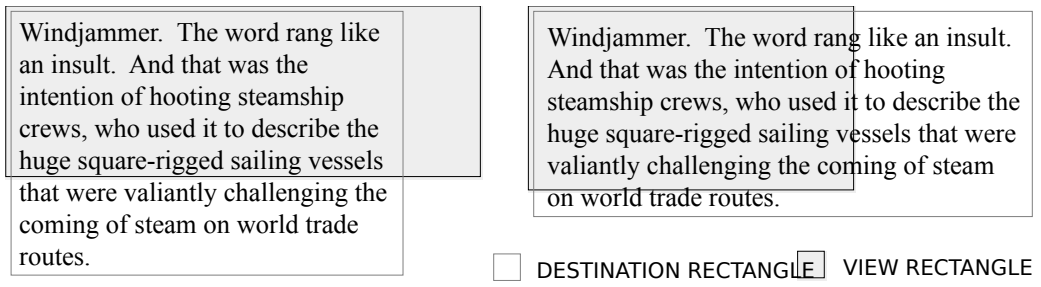

FIG 5 - DESTINATION AND VIEW RECTANGLES

Editing operations may shorten or lengthen the text. The bottom of the destination rectangle can extend to accommodate the end of the text. In other words, you can think of the destination rectangle as bottomless. The sides of the destination rectangle determine the beginning and the end of each line, and its top determines the position of the first line.

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.

selRect — The selection rectangle boundaries, in local coordinates.

lineHeight — The vertical spacing of lines of text. In a monostyled edit structure, the value specifies the fixed vertical distance from the ascent line of one line of text to the ascent line of the next line.

Multistyled Edit Structure. In a multistyled edit structure, this field is set to -1, which indicates that line heights are calculated independently for each line based on the maximum value for any individual character on that line.

fontAscent — The font ascent line. For monostyled text, the value specifies how far above the baseline the pen is positioned to begin drawing the caret or highlighting. (For single-spaced text, this is the height of the text in pixels.)

Multistyled Edit Structure. In a multistyled edit structure, this field is set to -1, which indicates that the font ascent is calculated independently for each line based on the maximum value for any individual character on that line.

selPoint — The point selected with the mouse, in the local coordinates of the current graphics port.

selStart — The byte offset of the beginning of the selection range. When you create an edit structure, TextEdit initialises this field to 0. (Byte offset 0 refers to the first byte in the text buffer.)

---

6 Note that the Dialog Manager makes the destination rectangle extend twice as far on the right as the view rectangle, so that horizontal scrolling can be used.

selEnd — The byte offset of the end of the selection range. (Note that, to include that byte, this value must be one greater than the position of the last byte offset of the text.) When you create an edit structure, TextEdit initialises this field to 0. With both selStart and selEnd initialised to 0, the insertion point is placed at the beginning of the text.

active — Set by TextEdit when an edit structure is activated using TEActivate and then reset when the edit structure is rendered inactive using TEDeactivate.

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 as long as the mouse button is held down within the text.

just — The type of text alignment (default, left, centre, or right).

teLength — The number of bytes in the text to be edited. The maximum allowable length is 32,767 bytes. When you create an edit structure, TextEdit initialises this field to 0.

hText — A handle to the text. When you create an edit structure, TextEdit initialises this field to point to a zero-length block in the heap.

hDispatchRec — Handle to the TextEdit dispatch structure. For internal use only.

clikStuff — TextEdit sets this field to reflect whether the most recent mouse-down event occurred on the leading or trailing edge of a glyph. Used internally by TextEdit to determine a caret position.

crOnly — Specifies whether or not text wraps at the right edge of the destination rectangle. If the value is positive, text does wrap. If the value is negative, new lines are specified explicitly by Return characters only and text does *not* wrap at the edge of the destination rectangle.

txFont — For a monostyled edit structure, this field specifies the font of all the text in the edit structure. If you change this value, the entire text of this edit structure has the new characteristic when it is redrawn. (If you change the value, you should also change the lineHeight and fontAscent fields as appropriate.)

Multistyled Edit Structure. In a multistyled edit 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 — For a monostyled edit structure, this field specifies the character attributes of all the text in an edit structure. If you change this value, the entire text of this edit structure has the new characteristic when it is redrawn. (If you change this value, you should also change the lineHeight and fontAscent fields as appropriate.)

Multistyled Edit 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 in the edit structure. If you change this value, the entire text in this edit structure has the new characteristic when it is redrawn.

txSize — In a monostyled edit structure, this field is set to the size of the text in points.

Multistyled Edit Structure. In a multistyled edit structure, this field is set to is -1, indicating that the edit 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 — Pointer to the graphics port associated with this edit structure.

highHook — Universal procedure pointer to the function which deals with text highlighting.

caretHook — Universal procedure pointer to the function that controls the appearance of the caret.

numLines — The number of lines in the text.

lineStarts — An array containing the character position of the first character in each line. It is declared to have 16001 elements to comply with Pascal range checking. This is a dynamic data structure having only as many elements as needed. TextEdit calculates these elements internally, so do not change the elements of this array. Because this data structure grows and shrinks, the size of the edit structure changes.

The Dispatch Structure

The hDispatchRec field of the edit structure stores a handle to the **dispatch structure**. The dispatch structure is an internal data structure whose fields, referred to as hook fields or hooks, contain the addresses of functions which TextEdit uses internally to, for example, measure and draw text or determine a character's position on a line. These functions, called **hook functions**, determine the way TextEdit behaves.[7]

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.

Initialising TextEdit

Before using TextEdit, you need to initialise TextEdit using TEInit which, amongst other things, sets up the private scrap and allocates a handle to it. You may also need to get information about the installed version of TextEdit using Gestalt with the selector gestaltTextEditVersion[8].

Creating, and Disposing of, a Monostyled Edit Structure

Creating a Monostyled Edit Structure

To use TextEdit functions, you must first create an edit structure using TENew. TENew returns a handle to the newly-created monostyled edit 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. To ensure that the first and last glyphs in each line are legible in a document window, you should

7 You can use a TextEdit customisation function to replace the address of a default hook function with the address of your own customised function.

8 Gestalt is described at Chapter 23 — Miscellany.

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. 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 an edit structure is created, TextEdit initialises the edit structure's fields based on values in the current colour graphics port structure and on the type of edit structure you create.

Disposing of an Edit Structure

Memory allocated for an edit structure may be released by calling TEDispose.

Setting the Text of an Edit Structure

When you create an edit structure, it does not contain any text until the user either enters text through the keyboard or opens an existing document. The following describes how to specify *existing* text to be edited.

There are two ways to specify existing text to be edited, namely, by using TESetText or by setting the hText field of the edit structure directly.

Calling TESetText

When a user opens a document, your application can bring the document's text into the text buffer of an edit structure by calling TESetText. TESetText creates a copy of the text and stores the copy in the existing handle of the edit structure's hText field.

One of the parameters you pass to TESetText specifies the length of the text. TESetText resets the teLength field of the edit structure with this value and uses it to determine the end of the text. It also sets the selStart and selEnd fields to the last byte offset of the text so that the insertion point is positioned at the end of the displayed text. Finally, TESetText calculates the line breaks, eliminating the necessity for your application to perform that task.

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

Changing the hText Field

The second method saves memory if you have a lot of text. In this method, you bring text into an edit structure by directly changing the hText field in the edit structure, replacing the existing handle with the handle of the new text. When you do this for a monostyled edit structure, you need to modify the teLength field to specify the length of the new text and then call TECalText to recalculate the lineStarts array and numLines values to match the new text.

Responding to Events

Activate Events

When your application receives an activate event, it must call TEActivate for an activate event and TEDeactivate for a deactivate event.

An application can have more than one edit structure associated with it. The active edit structure is the one where the next editing operation will take place. TEActivate visually identifies an edit structure as the active one by either highlighting the selection range or by displaying a caret at the insertion point. TEDeactivate changes an edit structure's status from active to inactive, removes the highlighting or caret and, if outline highlighting is enabled, frames the selection range or displays a gray, unblinking caret.

Typically, you include edit structure activation and deactivation in that function in your application which handles window activation and deactivation. That said, it is possible to modify the text of an edit structure associated with a background window; however, to do so, you need to call TEActivate for that edit structure before you call any other TextEdit functions.

Note that, when you use TEClick and TESetSelect (see below) to set the selection range or insertion point, the selection range is not highlighted and the blinking caret is not displayed until the edit structure is activated. (However, if outline highlighting is activated[9], the text of the selection range is framed or a gray, unblinking caret is displayed.)

Update Events — Calling TEUpdate

Your application needs to call TEUpdate every time the Event Manager reports an update event for a text editing window. In addition, you must call TEUpdate after changing any fields of the edit structure which affect the appearance of the text or after any editing or scrolling operation which alters the onscreen appearance of the text.

EraseRect and TEUpdate should be called after BeginUpdate and before EndUpdate. (If you do not include the EraseRect call, the black caret may sometimes remain visible (unblinking) when the window is deactivated.)

Mouse-Down Events — Calling TEClick

When your application receives notification of a mouse-down event that it determines TextEdit should handle, it must pass the event on to TEClick. TEClick tells TextEdit that a mouse-down event has occurred. Before calling TEClick, however, your application must perform the following steps:

- Convert the mouse location passed in the event structure from global coordinates to the local coordinates required by TEClick.
- Determine if the Shift key was held 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.

When TEClick is called, the clickTime field of the edit structure contains the time when TEClick was last called. When TEClick returns, it sets the clickTime field, adjusting the current tick count. The default click loop function uses this value.

Key-Down Events - Accepting Text Input

When your application receives a key-down event which it determines TextEdit should handle, it must call TEKey to accept the keyboard input a byte at a time (or to delete a

9 Outline highlighting may be activated and deactivated using TEFeatureFlag.

character when the user backspaces over it). 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.

Null Events - Caret Blinking

To force the insertion point caret to blink, your application must call TEIdle whenever it receives a null event. You must also ensure that the sleep parameter in the WaitNextEvent call is set to a value equal to or less than the value stored in the low-memory global CaretTime, which determines the blinking time for the caret[10]. That value can be retrieved by a call to LMGetCaretTime.

If there is more than one edit structure associated with an active window, you must ensure that you pass TEIdle the handle to the currently active edit 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 a single edit structure, between edit structures, or across applications. The relevant functions, and their effect in the case of a monostyled edit 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 edit structure. (Used for monostyled text only.)
TEToScrap	Copy TextEdit private scrap to the Scrap Manager's desk scrap.	Copying via the Scrap Manager's desk scrap is required if monostyled text is to be carried across applications.
TEFromScrap	Copy the Scrap Manager's desk scrap to TextEdit private scrap.	Copying via the Scrap Manager's desk scrap is required if monostyled text is to be carried across applications.
TEGetScrapLength	Determine the length of the monostyled text to be pasted.	Returns the size, in bytes, of the text in the private scrap.

If you are using TEFromScrap to support pasting to your application from the desk scrap, you will need to ensure that a paste will not cause the TextEdit limit of 32,767 bytes to be exceeded. One way to do this is to call the Scrap Manager's GetScrap function to get the size of the text to be pasted, add this to the size of the text in the edit structure, subtract the size of the selection range, and then compare the result against the maximum allowable length of the edit structure.

You will need to call your vertical scroll bar adjustment function immediately after cut and paste operations.

10 The blinking time is set by the user using the General Controls Control Panel.

Inserting and Deleting Text

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

Function	Use To	Comments
TEInsert	Insert monostyled text into the edit structure immediately before the selection range or insertion point.	Does not affect the selection range. Redraws the text if necessary. Use for monostyled text only.
TEDelete	Remove the selected range of text from the edit structure.	Does not transfer the text to either TextEdit's private scrap or the Scrap Manager's desk 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

You can use TESetSelect to specify the selection range or the position of the insertion point as determined by the application. For example, you can use TESetSelect to position the caret at the start of a data entry field where you want the user to enter a value. TESetSelect modifies the selStart and selEnd fields of the edit structure.

To select a range of text, you pass TESetSelect the handle to the edit structure along with the byte offsets of the starting and ending characters. You can set the selection range (or insertion point) to any character position corresponding to byte offsets 0 to 32767. To display a caret at the insertion point, specify the same values for the selStart and selEnd parameters.

To implement a Select All menu command, specify 0 for starting offset parameter and use the teLength field of the edit structure for the ending offset 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 function does not adjust the scroll bars as the text is scrolled, a situation which can be overcome by replacing the default click loop function with an application-defined click loop function which updates the scroll bars as it scrolls the text.

The clickLoop field of the edit structure contains a universal procedure pointer to a click loop 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 edit structure's clickLoop field.

Scrolling Text

To scroll the text when a mouse-down event occurs in a scroll bar, your application needs to first determine how far to scroll the text. The basic value for vertical scrolling of a monostyled edit structure is typically the value in the lineHeight field of the edit 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, this value is typically multiplied by the number of text lines in the view rectangle minus 1. Scrolling by dragging the scroll box 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.

To scroll the text, you can use either TEScroll or TEPinScroll, specifying the number of pixels to scroll. The only difference between these two functions is that TEPinScroll 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 an edit structure by calling TESetAlignment (old name TESetJust). 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 InvalRect 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 16 — Files demonstrates opening and saving monostyled TextEdit documents.

Multistyled TextEdit

This section addresses additional factors and considerations applying to multistyled TextEdit.

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

Text which 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 edit 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 edit structure and the dispatch structure are the only data structures associated with monostyled text. However, when you allocate a multistyled edit structure, a number of additional subsidiary data structures are created to support the text styling capabilities. The additional data structures associated with a multistyled edit structure are shown at Fig 6.

The Style Structure

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 edit structure is created, the bytes at the txFace, txFace, and filler fields of the edit structure contain a handle to the style structure.) The remaining additional data structures are, in fact, elements of the style structure. Those elements are as follows:

- A handle to a **style table**, which has one entry for each distinct set of character attributes in the edit structure. Each element in the style table is a **style table element structure**.

- A handle to the **line-height table**, which provides vertical spacing and line ascent information for the text to be edited. The line-height table comprises one **line-height element structure** for each line of an edit structure. A line number is a direct index into the array of line-height element structures.

- A **style run table**, which is an array of **style run structures**, each of which provides, for each style run, the offset of the starting character to which the character attributes stored in the style table apply and an index into the style table.

- A handle to the **null style structure**, which contains a handle to the **style scrap structure**. The style scrap structure, which is part of the style scrap, stores character attribute information associated with a null selection to be applied to inserted text. It also holds character attribute information associated with a selected range of multistyled text when the character attributes are to be copied, or the text and its attributes are to be cut and copied. Part of the style scrap structure is the **scrap style table**, which comprises one **scrap style element structure** for each style run in the style scrap structure. Scrap style element structures hold character attribute information similar to that contained in the style table element structure.

STYLE STRUCT

```
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;
```

STYLE TABLE ELEM

```
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;
```

LINE-HEIGHT TABLE EL MENT

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

STYLE RUN TABLE ELE MENT

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

NULL STYLE STRUC

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

STYLE SCRAP STRUC URE

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

SCRAP STYLE TABLE ELEME

```
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

Creating a Multistyled Edit Structure

The multistyled edit structure is created by calling TEStyleNew.

Setting the Text

The alternative method of setting the text (that is, directly setting the hText field of the edit structure) is somewhat more cumbersome for a multistyled text because TECalText does not update the style run table properly. To compensate for this, your application needs to perform the following tasks:

- Before changing the edit structure's hText field, reduce the style run table to one entry. Do this by setting the edit structure's selStart field to 0 and the selEnd field to 32,767, and then call TESetStyle.
- Before calling TECalText, set the start character (startChar) field of the style run table to the length of the new text plus one.

TEKey and Multistyled Text

When the user backspaces over characters in a multistyled edit structure, TEKey deletes the characters (as in a monostyled edit structure) but also saves the character attributes associated with the last character deleted in order to apply it to any new characters the user enters. The character attributes are saved in the null scrap's style scrap structure. As soon as the user clicks in another area of the text, TEKey clears the attributes from the null scrap.

Cutting, Copying, Pasting, Inserting, and Deleting Text

The following shows the effects of TECut and TECopy when multistyled text is involved. It also describes TEStylePaste, which is used for pasting multistyled text, and the additional function (TENumStyles), which is involved in cutting and copying multistyled text:

Function	Use To	Comments
TECut	Cut text.	For multistyled text:
TECopy	Copies text.	Copies both the text and its character attribute information to the Scrap Manager's desk scrap under scrap types 'TEXT' and 'styl'. Copies the text to the TextEdit private scrap and the attributes stored in the style table to the TextEdit style scrap.
TEStylePaste	Paste multistyled text.	Pastes both the text and its attributes from the Scrap Manager's desk scrap to the edit structure. (Use the Scrap Manager function GetScrap to check the size of the text ('TEXT' data) to be pasted, passing NULL for the hDest parameter to avoid copying the data.)
TENumStyles	Determine the memory required for the style scrap before cutting or copying multistyled text.	TENumStyles returns the number of attribute changes contained in the range of text. Multiply this by sizeof(ScrpSTElement) and add two to get the number of bytes required.

The following describes TEStyleInsert, which is used to insert multistyled text, and the additional effects of TEDelete when used to delete multistyled text:

Function	Use To	Comments
TEStyleInsert	Insert multistyled text into the edit structure immediately before the	Does not affect the selection range. Redraws the text if necessary. Applies the specified character attributes to the

	selection range or insertion point.	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.)
TEDelete	Remove the selected range of text from the edit structure.	Does not transfer the text to either TextEdit's private scrap or the Scrap Manager's desk scrap. Redraws the remaining text if necessary. For multistyled text, the character attributes are saved in the null scrap to be applied to characters after the text has been deleted. When the user clicks in some other area of the text, the attributes are removed from the null scrap. TEDelete can be used to implement a Clear command.

Scrolling Text

The number of pixels to be scrolled vertically for each line of text to be scrolled (determined from the lineHeight field in a monostyled edit structure) is determined from the lnHeight field of the line height table in multistyled edit structures.

Setting and Checking Text Attributes

Your application may need to check the current attributes of a range of text to determine which ones are consistent across a range of text. Your application may also need to manipulate the font, style, size, and colour of a range of text, the text selection consisting of the entire text of the edit structure, a segment of text, a single character, or even an insertion point. The following functions are relevant in this regard:

Function	Use To	Comments
TESetStyle	Change the font, size, style and/or colour of the text in the selection range.	Typically used to implement menu commands relating to modifying text attributes. If called for an insertion point, TextEdit stores the specified attribute information in the null scrap's style scrap structure.
TEContinuousStyle	Examine the current selection range and determine whether a specified style attribute is continuous across the range.	Can be used as an aid in toggling styles or to determine which of the items in a Style menu should have a checkmark. The mode parameter specifies which attributes of the selected text are to be examined.

Checking Attributes in a Selection Range

The following example application-defined function shows how to determine the font, size, style and colour of the current selection range.

```
void  doGetCurrentSelection(TextStyle *textStyleRec,TEHandle editRecHdl)
{
   SInt16  mode;
   Boolean    continuous;

   // doFont, doFace, doSize, and doColor are constants which specify font family number,
   // character style, type size, and colour

   mode = doFont + doFace + doSize + doColor;
   continuous = TEContinuousStyle(&mode,textStyleRec,editRecHdl);

   // When TEContinuousStyle returns, each bit in mode that was set on entry will have
   // been cleared if that style element was not continuous.  For those attributes which
   // were continuous, the text style (TextStyle) structure fields will contain the
   // actual values.
```

```
    if((mode & doFont) != 0)
        // Font for selection = tsFont field of the TextStyle structure.
    else
        // More than one font in selection.

    if((mode & doFace) != 0)
        // tsFace field of the TextStyle structure contains the styles (or plain) common
        // to the selection.
    else
        // No text face is common to the entire selection.

    if((mode & doSize) != 0)
        // Size for selection = tsSize field of the TextStyle structure.
    else
        // More than one size in selection.

    if((mode & doColor) != 0)
        // Color for selection = tsColor field of the TextStyle structure.
    else
        // More than one colour in selection.
}
```

Handling a Font Menu

The following example application-defined function shows how to handle a Font menu item selection.

```
void  doFontMenu(WindowPtr windowPtr,TEHandle editRecHdl,SInt16 menuItem)
{
   TextStyle  styleRec;
   Str255     fontName;
   SInt16     fontID;

   GetMenuItemText(GetMenuHandle(mFont),menuItem,fontName);    // Get text of menu item.
   GetFNum(fontName,&fontID);                                  // Get font number matching font name.
   styleRec.tsFont = fontID;                    // Assign to tsFont field of text style structure.
   TESetStyle(doFont,&styleRec,true,editRecHdl);// Apply style to selected text ...
                                                                // and redraw text immediately.
   doAdjustScrollBars(windowPtr,false);                    // Adjust scroll bars.
}
```

Handling a Size Menu

The following example application-defined function shows how to handle a Size menu item selection.

```
void  doSizeMenu(WindowPtr windowPtr,TEHandle editRecHdl,SInt16 menuItem)
{
   SInt16     sizeChosen;
   TextStyle  styleRec;

   doGetSize(GetMenuHandle(mSize),menuItem,sizeChosen);  // Get size from menu item.
   styleRec.tsSize = sizeChosen;                                  // Assign to tsSize field of text
                                                                  // style structure.
   TESetStyle(sizeChosen,&styleRec,true,editRecHdl);  // Apply size to selected text ...
                                                                  // and redraw text immediately.
   doAdjustScrollBars(windowPtr,false);                      // Adjust scroll bars.
}
```

Handling a Style Menu

The following example application-defined function shows how to handle a Style menu item selection.

```
void  doHandleStyleMenu(WindowPtr windowPtr,TEHandle editRecHdl,SInt16 menuItem)
{
   TextStyle  styleRec;

   switch menuItem
   {
      case iPlain:
         styleRec.tsFace = (Style) normal;
         break;
```

```
            case iBold:
               styleRec.tsFace = (Style) bold;
               break;

            case iItalic:
               styleRec.tsFace = (Style) italic;
               break;

            case iUnderline:
               styleRec.tsFace = (Style) underline;
               break;

            case iOutline:
               styleRec.tsFace = (Style) outline;
               break;

            case iShadow:
               styleRec.tsFace = (Style) shadow;
               break;
         }

         if(menuItem != 1)                                                  // If Plain not selected
            TESetStyle(doFace + doToggle,&styleRec,true,editRecHdl)      // ... include doToggle.
         else                                                                  // If Plain selected
            TESetStyle(doFace,&styleRec,true,editRecHdl);                  // ... delete doToggle.

         doAdjustScrollBars(windowPtr,false);
      }
```

Note that, if you call TESetStyle with the value of false for the redraw parameter, TextEdit does not redraw the text or recalculate line breaks, line heights or font ascents until the next update occurs. This will cause problems if you call a function that uses any of this information before the update occurs.

The following example application-defined function checks the character attributes of the current selection range and, for each style that is continuous across the range, marks the item in the Style menu.

```
void  doAdjustStyleMenu(TEHandle editRecHdl)
{
   MenuHandle styleMenuHdl;
   TextStyle     styleRec;
   SInt16        mode;

   mode = doFace;
   styleMenuHdl = GetMenuHandle(mStyle);

   // If TEContinuousStyle returns true, there is at least one style that is continuous
   // over the selection.  Note that it might be plain, which is the absence of styles.

   if(TEContinuousStyle(&mode,&styleRec,editRecHdl))                  // There is a
   {                                                                        // continuous
      CheckMenuItem (styleMenuHdl,iPlain,styleRec.tsFace == normal);  // style so mark
      CheckMenuItem (styleMenuHdl,iBold,styleRec.tsFace & bold);         // all menu items
      CheckMenuItem (styleMenuHdl,iItalic,styleRec.tsFace & italic);    // appropriately.
      // Set other items as appropriate.
   }
   else                                                                     // There is no continuous
   {                                                                           // style so unmark all
      CheckMenuItem (styleMenuHdl,iPlain,false);                     // menu items.
      CheckMenuItem (styleMenuHdl,iBold,false);
      CheckMenuItem (styleMenuHdl,iItalic,false);
      // Set other items as appropriate.
   }
}
```

Handling the Undo Command

If you are implementing an Undo command for multistyled text, you need to save the character attribute information along with the text. There are a number of ways to do this. For example, when you want to save the current attributes of the selected text to allow the user to revert to them, your application calls TEGetStyleScrapHandle, which returns a

handle to the style scrap's style structure containing the attributes used for the selected text.

To restore the style later, you call TEUseStyleScrap. You also need to save the offsets into the edit structure's text buffer of the first and last characters to which the character attribute information is applied.

Saving and Opening Multistyled TextEdit Documents

Saving a Multistyled TextEdit Document

To save the contents of a document created with a multistyled edit structure, you need to save all the associated character attribute information[11] in addition to the text. Because the text attribute information in the style scrap is easier to export that the style structure itself (because it uses the Desk Manager's 'styl' format), you should use the TextEdit functions that use the style scrap for moving character attribute information (TEGetStyleScrapHandle and TEUseStyleScrap). For example, you can use the following steps to save a multistyled text document to disk:

- Create a text file, select all the text of the edit structure, and save it to the data fork.
- Call TEGetStyleScrapHandle to get a handle to the style scrap structure. This creates the style scrap structure and uses it to store the character attribute information.
- Save the character attribute information in the resource fork of the file.

The following example application-defined function uses this method.

```
void doSaveAsTextEdit(TEHandle editRecHdl)
{
StandardFileFileReply   fileReply;
StScrpHandle            styleScrapHdl;
SInt32                  dataLength;
SInt16                  dataRefNum;
SInt16                  rsrcRefNum;
SInt16                  savedStart;
SInt16                  savedEnd;
OSErr                      osError;
Handle                  editText;

StandardPutFile("\pSave as:","\pUntitled",&fileReply);
if(fileReply.sfGood)
{
   savedStart = (*editRecHdl)->selStart;                    // Save current selection ...
   savedEnd = (*editRecHdl)->selEnd;                           // starting and ending offsets.

   (*editRecHdl)->selStart = 0;                             // Select all text. (Do not ..
   (*editRecHdl)->selEnd = (*editRecHdl)->teLength;  // use TESetSelect.)
   styleScrapHdl == GetStylScrap(editRecHdl);               // Get list of all attributes.
   (*editRecHdl)->selStart = savedStart;                    // Reset original selection.
   (*editRecHdl)->selEnd = savedEnd;

   if(!(fileReply.sfReplacing))                          // Create file & resource ...
   {                                                                  // fork if not already created.
      osError = FSpCreate(&fileReply.sfFile,'K JB','TEXT',fileReply.sfScript);
      FSpCreateResFile(fileReply,sfFile,'K JB','TEXT',fileReply.sfScript);
      osError = ResError();
   };

   osError = FSpOpenDF(&fileReply.sfFile,fsCurPerm,&dataRefNum);   // Open data fork ..
   rsrcRefNum = FSpOpenResFile(&fileReply.sfFile,fsCurPerm);       // and resource fork.
   osError = ResError();

   dataLength = (*editRecHdl)->teLength;                        // Write text.
   editText = (*editRecHdl)->hText;
   osError = FSWrite(dataRefNum,&dataLength,*editText);
```

11 For the font, remember to save the font name, not the font number.

```
        AddResource((Handle) styleScrapHdl,'styl',0,"\p");    // Write attributes.
        WriteResource((Handle) styleScrapHdl);
        ReleaseResource((Handle) styleScrapHdl);

        osError = FSClose(dataRefNum);                              // Close data fork ...
        CloseResFile(rsrcRefNum);                                       // and resource fork.
        osError = ResError();
    }
}
```

Notice that this function avoids using TESetSelect to select all of the edit structure's text. TESetSelect sets *and* highlights the selection range that you specify. You do not want the text to be highlighted because this could mislead the user into presuming that some other action is required.[12]

Opening a Multistyled TextEdit Document

The following example application-defined function shows how to open a multistyled TextEdit document:

```
void doOpenAsTextEdit(TEHandle editRecHdl)
{
    StandardFileFileReply   fileReply;
    SFFileTypes                 fileTypes;
    SInt16                      dataRefNum;
    SInt16                      rsrcRefNum;
    Handle                      textBuffer;
    SInt32                      textLength;
    StScrpHandle                styleScrapHdl;
    OSErr                           osError;
    UInt8                           savedState;

    fileTypes[0] = 'TEXT';

    StandardGetFile(NULL,1,fileTypes,&fileReply);
    if(fileReply.sfGood)
    {
        osError = FSpOpenDF(&fileReply.sfFile,fsCurPerm,&dataRefNum);
        osError = SetFPos(dataRefNum,fsFromStart,0);
        osError = GetEOF(dataRefNum,&textLength);

        if(textlength > 32767)
            textLength = 32767;

        textBuffer = NewHandle((Size) textLength);                  // Allocate buffer for text.
        osError = FSRead(dataRefNum,&textLength,*textBuffer); // Read text to buffer.
        LockHHi(textBuffer);
        TESetText(*buffer,textlength,editRecHdl);                       // Put text in edit structure.
        HUnlock(textBuffer);
        DisposeHandle(textBuffer);                                          // Get rid of buffer.

        osError = FSClose(dataRefNum);                                      // Close data fork.

        rsrcRefNum := FSpOpenResFile(&fileReply.sfFile,fsCurPerm);   // Open resource fork.
        osError = ResError();

        styleScrapHdl = GetResource('styl',0);                      // Get style scrap.
        osError = ResError();
        if(styleScrapHdl != NULL)
        {                                                                               // Apply attributes ...
            savedState = HGetState((Handle) styleScrapHdl);         // to edit structure
            TEUseStyleScrap(0,textLength,styleScrapHdl,true,editRecHdl);
            HSetState((Handle) styleScrapHdl,savedState);
        }

        CloseResFile(rsrcRefNum);                                           // Close resource fork.
        osError = ResError();
    }
}
```

12 However, if you want to use TESetSelect, you can circumvent highlighting of the selection range if you first render the edit structure inactive. Also, if you have the outline highlighting feature turned on, turn it off.

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 provided by the system software which allow you to specify strings for various purposes, sort strings, convert case or strip diacritical marks from text for sorting purposes, search and replace text, find word boundaries and line breaks when laying out lines of text, and format numbers, currency, dates, and times. The following is concerned only with the latter, that is, formatting 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, numbers, and the preferred unit of measurement. 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 structure.	Long date-time value.
LongSecondsToDate	Long date-time value.	Long date 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 these fields. 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	A fatal error occurred during the parse.
dateTimeNotFound	A valid date or time value could not be found in the string.
sepNotIntlSep	A valid date or time value was found; however, 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

When you present numbers to the user, or when the user enters numbers for your application to use, you need to convert between the internal numeric representation of the number and the output (or input) format of the number. The Text Utilities provide several functions for performing these conversions.

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

If you are working with floating point numbers, or if you want to accommodate the possible differences in number output formats for different countries and regions of the

world, you need to work with **number format specification strings**. Number format specification strings define the appearance of numeric strings in your application.

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 specifying 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	123456.78	123,456.78
###,###.0##,###	1234	1,234.0
###,###.0##,###	3.141592	3.141,592
###;(000);^^^	-1	(001)
###.###	1.234999	1.235
###'CR';###'DB';''zero''	1	1CR
###'CR';###'DB';''zero''	0	'zero'
##%	0.1	10%

The number formatting functions always fill in integer digits from the right and decimal places on 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***
###'my'###	1	1
###'my'###	123	123
###'my'###	1234	1my1234
0.###'my'###	0.1	0.1

0.###'my'###	0.123	1.123
0.###'my'###	0.1234	0.123my4

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 too many decimal places, the decimal portion is automatically rounded. For example, given the format ###.###, a value of 1234.56789 results in an error condition, and a value of 1.234999 results in the rounded-off value 1.235.

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.

Functions	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[13], in the format specification string.[14] 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.

Converting Between Floating Point Numbers and Numeric Strings

Once you have a NumFormatString structure which defines the format of numbers for a certain region of the world, you can convert floating point numbers into numeric strings and numeric strings into floating point numbers using the following functions:

Functions	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:

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

14 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.

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
```

Values for TESetStyle

```
doFont                  = 1
doFace                  = 2
doSize                  = 4
doColor                 = 8
doAll                   = 15
addSize                 = 16
```

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;
```

Edit 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 structure.
    short               fontAscent;     // Font ascent. -1 in multistyled edit 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 edit structure.
    GrafPtr         inPort;             // Pointer to grafPort for this edit 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;
```

Style Table Element

```
struct STElement
{
    short           stCount;            // Number of runs in this style.
    short           stHeight;           // Line height.
    short           stAscent;           // Font ascent.
    short           stFont;             // Font (family) number.
    StyleField      stFace;             // Character Style.
    short           stSize;             // Size in points.
    RGBColor        stColor;            // absolute (RGB) color.
};
typedef struct STElement STElement;
typedef STElement TEStyleTable[1777];
typedef STElement *STPtr;
typedef STPtr *STHandle;
```

Line Height Table Element

```
struct LHElement
{
    short           lhHeight;           // Maximum height in line.
    short           lhAscent;           // Maximum ascent in line.
};
typedef struct LHElement LHElement;
typedef LHElement LHTable[8001];
typedef LHElement *LHPtr;
typedef LHPtr *LHHandle;
```

Style Run Table Element

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

Null Style Structure

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

```
typedef struct NullStRec NullStRec;
typedef NullStRec *NullStPtr;
typedef NullStPtr *NullStHandle;
```

Style Scrap Structure

```
struct StScrpRec
{
    short           scrpNStyles;        // Number of styles in scrap.
    ScrpSTTable     scrpStyleTab;       // Table of styles for scrap.
};
typedef struct StScrpRec StScrpRec;
typedef StScrpRec *StScrpPtr;
typedef StScrpPtr *StScrpHandle;
```

Scrap Style Table Element

```
struct ScrpSTElement
{
    long            scrpStartChar;      // Starting character position.
    short           scrpHeight;         // Character height.
    short           scrpAscent;         // Ascent above baseline this font.
    short           scrpFont;           // Font (family) number.
    StyleField      scrpFace;           // Character style.
    short           scrpSize;           // Font size (points).
    RGBColor        scrpColor;          // Colour used this style.
};
typedef struct ScrpSTElement ScrpSTElement;
typedef ScrpSTElement ScrpSTTable[1601]; //  ARRAY [0..1600] OF ScrpSTElement.
```

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

Initialising TextEdit, Creating Edit Structure, Disposing of Edit Structure

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

Activating and Deactivating an Edit Structure

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

Setting and Getting an Edit Structure's Text and Character Attribute Information

```
void            TEKey(short key,TEHandle hTE);
void            TESetText(const void *text,long length,TEHandle hTE);
CharsHandle     TEGetText(TEHandle hTE);
void            TESetStyleHandle(TEStyleHandle theHandle,TEHandle hTE);
TEStyleHandle   TEGetStyleHandle(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 an Edit Structure

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

Managing the TextEdit Private Scrap

```
#define         TEScrapHandle(); (* (Handle*) 0xAB4);
#define         TEGetScrapLength(); ((long) * (unsigned short *) 0x0AB0);
void            TESetScrapLength(long length);;
```

Checking, Setting, and Replacing Styles

```
void            TESetStyle(short mode,const TextStyle *newStyle,Boolean redraw,TEHandle hTE);
void            TEReplaceStyle(short mode,const TextStyle *oldStyle,const TextStyle *newStyle,  Boolean
                fRedraw,TEHandle hTE);
Boolean         TEContinuousStyle(short *mode,TextStyle *aStyle,TEHandle hTE);
void            TEStyleInsert(const void *text,long length,StScrpHandle hST,TEHandle hTE);
TEStyleHandle   TEGetStyleHandle(TEHandle hTE);
StScrpHandle    TEGetStyleScrapHandle(TEHandle hTE);
void            TEUseStyleScrap(long rangeStart,long rangeEnd,StScrpHandle newStyles,Boolean fRredraw,TEHandle
                hTE);
long            TENumStyles(long rangeStart,long rangeEnd,TEHandle hTE);
```

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 Text1

```
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
// Text1.c
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
//
// 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 value, and scroll bar 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:
//
// •   An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit, and Help dialog
//     pop-up menus (preload, non-purgeable).
//
// •   A 'WIND' resource (purgeable) (initially visible).
//
// •   'CNTL' resources (purgeable) for the vertical scroll bars in the text editor window
//     and Help dialog, and for the pop-up menu in the Help Dialog.
//
// •   A 'DLOG' resource (purgeable, initially invisible) and associated 'dctb' resource
//     (purgeable) for the Help dialog.
//
// •   'DITL' resources (purgeable) for the 'ALRT' and 'DLOG' resources.
//
```

```
// •   '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, doesActivateOnFGSwitch, and
//      is32BitCompatible    flags set.
//
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

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

#include <Appearance.h>
#include <Balloons.h>
#include <ControlDefinitions.h>
#include <Devices.h>
#include <Gestalt.h>
#include <LowMem.h>
#include <Scrap.h>
#include <Sound.h>
#include <StandardFile.h>
#include <ToolUtils.h>

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

#define rMenubar            128
#define mApple                  128
#define  iAbout             1
#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 rWindow                 128
#define rVScrollbar         128
#define rErrorStrings       128
#define  eMenuBar               1
#define  eMenu                  2
#define  eWindow                3
#define  eDocStructure      4
#define  eEditRecord        5
#define  eExceedChara           6
#define  eNoSpaceCut        7
#define  eNoSpacePaste      8
#define kMaxTELength        32767
#define kTab                    0x09
#define kDel                0x7F
#define kReturn                 0x0D
#define topLeft(r)        (((Point *) &(r))[0])
#define botRight(r)         (((Point *) &(r))[1])

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

typedef struct
{
    TEHandle         editRecHdl;
    ControlHandle vScrollbarHdl;
} docStructure, **docStructureHandle;

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

ControlActionUPP scrollActionFunctionUPP;
```

```
TEClickLoopUPP        customClickLoopUPP;
Boolean                  gMacOS85Present = false;
Boolean                  gDone;
Boolean                  gInBackground;
RgnHandle                gCursorRegion;
SInt16                gNumberOfWindows = 0;
SInt16                gOldControlValue;

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

void              main                  (void);
void              doInitManagers      (void);
void              eventLoop             (void);
void              doIdle                (void);
void              doEvents              (EventRecord *);
void              doKeyEvent            (SInt8);
pascal void     scrollActionFunction (ControlHandle,SInt16);
void              doInContent           (EventRecord *);
void              doUpdate              (EventRecord *);
void              doActivate            (EventRecord *);
void              doActivateDocWindow   (WindowPtr,Boolean);
void              doOSEvent               (EventRecord *);
WindowPtr           doNewDocWindow          (void);
pascal Boolean customClickLoop      (void);
void              doSetScrollBarValue (ControlHandle,SInt16 *);
void              doAdjustMenus         (void);
void              doMenuChoice          (SInt32);
void              doFileMenu            (SInt16);
void              doEditMenu            (SInt16);
SInt16            doGetSelectLength     (TEHandle);
void              doAdjustScrollbar   (WindowPtr);
void              doAdjustCursor      (WindowPtr);
void              doCloseWindow           (WindowPtr);
void              doSaveAsFile          (TEHandle);
void              doOpenCommand           (void);
void              doOpenFile            (FSSpec);
void              doDrawDataPanel       (WindowPtr);
void              doErrorAlert        (SInt16);
void              doHelpMenu            (SInt16);

extern void       doHelp              (void);

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

void  main(void)
{
  OSErr          osError;
  SInt32       response;
  Handle       menubarHdl;
  MenuHandle menuHdl;
  OSErr          osErr;

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

  doInitManagers();

  // ............................................................................................................... check whether Mac OS 8.5 or later is
present

  osError = Gestalt(gestaltSystemVersion,&response);

  if(osError == noErr && response >= 0x00000850)
    gMacOS85Present = true;

  // ...................................................................................................................................................... create
routine descriptors

  scrollActionFunctionUPP = NewControlActionProc((ProcPtr) scrollActionFunction);
  customClickLoopUPP = NewTEClickLoopProc((ProcPtr) customClickLoop);

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

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
```

```
    doErrorAlert(eMenuBar);
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  menuHdl = GetMenuHandle(mApple);
  if(menuHdl == NULL)
    doErrorAlert(eMenu);
  else
    AppendResMenu(menuHdl,'DRVR');

  osErr = HMGetHelpMenuHandle(&menuHdl);
  if(osErr == noErr)
    AppendMenu(menuHdl,"\pText1 Help");
  else
    doErrorAlert(eMenu);

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

  doNewDocWindow();

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

  eventLoop();
}

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

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

  InitGraf(&qd.thePort);
  InitFonts();
  InitWindows();
  InitMenus();
  TEInit();
  InitDialogs(NULL);
  InitCursor();

  FlushEvents(everyEvent,0);

  RegisterAppearanceClient();
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ eventLoop

void  eventLoop(void)
{
  EventRecord eventStructure;
  Boolean        gotEvent;
  SInt32       sleepTime;

  gDone = false;
  gCursorRegion = NewRgn();
  sleepTime = LMGetCaretTime();

  while(!gDone)
  {
    gotEvent = WaitNextEvent(everyEvent,&eventStructure,sleepTime,gCursorRegion);

    if(gotEvent)
      doEvents(&eventStructure);
    else
    {
      if(gNumberOfWindows > 0)
        doIdle();
    }
  }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doIdle

void  doIdle(void)
{
  docStructureHandle  docStrucHdl;
```

```
    WindowPtr            windowPtr;

    windowPtr = FrontWindow();

    docStrucHdl = (docStructureHandle) (GetWRefCon(windowPtr));;
    if(docStrucHdl != NULL)
        TEIdle((*docStrucHdl)->editRecHdl);
}

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

void  doEvents(EventRecord *eventStrucPtr)
{
    WindowPtr    windowPtr;
    SInt16       partCode;
    SInt8        charCode;

    switch(eventStrucPtr->what)
    {
        case mouseDown:
            partCode = FindWindow(eventStrucPtr->where,&windowPtr);
            switch(partCode)
            {
                case inMenuBar:
                    doAdjustMenus();
                    doMenuChoice(MenuSelect(eventStrucPtr->where));
                    break;

                case inContent:
                    if(windowPtr != FrontWindow())
                        SelectWindow(windowPtr);
                    else
                        doInContent(eventStrucPtr);
                    break;

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

                case inGoAway:
                    if(TrackGoAway(windowPtr,eventStrucPtr->where))
                        doCloseWindow(FrontWindow());
                    break;
            }
            break;

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

        case updateEvt:
            doUpdate(eventStrucPtr);
            break;

        case activateEvt:
            doActivate(eventStrucPtr);
            break;

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

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doKeyEvent

void  doKeyEvent(SInt8 charCode)
{
    WindowPtr                   windowPtr;
    docStructureHandle  docStrucHdl;
```

```
  TEHandle          editRecHdl;
  SInt16            selectionLength;

  windowPtr = FrontWindow();
  docStrucHdl = (docStructureHandle) (GetWRefCon(windowPtr));;
  editRecHdl = (*docStrucHdl)->editRecHdl;

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

  doDrawDataPanel(windowPtr);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ scrollActionFunction

pascal void  scrollActionFunction(ControlHandle controlHdl,SInt16 partCode)
{
  WindowPtr           windowPtr;
  docStructureHandle  docStrucHdl;
  TEHandle            editRecHdl;
  SInt16              linesToScroll;
  SInt16              controlValue, controlMax;

  windowPtr = (*controlHdl)->contrlOwner;
  docStrucHdl = (docStructureHandle) (GetWRefCon(windowPtr));;
  editRecHdl = (*docStrucHdl)->editRecHdl;

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

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

        case kControlPageUpPart:
        case kControlPageDownPart:
          linesToScroll = (((*editRecHdl)->viewRect.bottom - (*editRecHdl)->viewRect.top) /
                                    (*editRecHdl)->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(controlHdl,linesToScroll);
```

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

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

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doInContent

void  doInContent(EventRecord *eventStrucPtr)
{
  WindowPtr           windowPtr;
  docStructureHandle  docStrucHdl;
  TEHandle            editRecHdl;
  Point               mouseXY;
  ControlHandle       controlHdl;
  SInt16              partCode;
  Boolean             shiftKeyPosition = false;

  windowPtr = FrontWindow();
  docStrucHdl = (docStructureHandle) GetWRefCon(windowPtr);
  editRecHdl = (*docStrucHdl)->editRecHdl;

  mouseXY = eventStrucPtr->where;
  SetPort(windowPtr);
  GlobalToLocal(&mouseXY);

  if((partCode = FindControl(mouseXY,windowPtr,&controlHdl)) != 0)
  {
    gOldControlValue = GetControlValue(controlHdl);
    TrackControl(controlHdl,mouseXY,scrollActionFunctionUPP);
  }
  else if(PtInRect(mouseXY,&(*editRecHdl)->viewRect))
  {
    if((eventStrucPtr->modifiers & shiftKey) != 0)
      shiftKeyPosition = true;
    TEClick(mouseXY,shiftKeyPosition,editRecHdl);
  }
}

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

void  doUpdate(EventRecord *eventStrucPtr)
{
  WindowPtr           windowPtr;
  docStructureHandle  docStrucHdl;
  TEHandle            editRecHdl;
  GrafPtr             oldPort;

  windowPtr = (WindowPtr) eventStrucPtr->message;
  docStrucHdl = (docStructureHandle) (GetWRefCon(windowPtr));;
  editRecHdl = (*docStrucHdl)->editRecHdl;

  GetPort(&oldPort);
  SetPort(windowPtr);

  BeginUpdate((WindowPtr)eventStrucPtr->message);

  TEUpdate(&windowPtr->portRect,editRecHdl);
  UpdateControls(windowPtr,windowPtr->visRgn);

  doDrawDataPanel(windowPtr);

  EndUpdate((WindowPtr)eventStrucPtr->message);

  SetPort(oldPort);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doActivate
```

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

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

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doActivateDocWindow

void  doActivateDocWindow(WindowPtr windowPtr,Boolean becomingActive)
{
   docStructureHandle  docStrucHdl;
   TEHandle            editRecHdl;

   docStrucHdl = (docStructureHandle) (GetWRefCon(windowPtr));;
   editRecHdl = (*docStrucHdl)->editRecHdl;

   if(becomingActive)
   {
      SetPort(windowPtr);

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

      TEActivate(editRecHdl);
      ActivateControl((*docStrucHdl)->vScrollbarHdl);
      doAdjustScrollbar(windowPtr);
   }
   else
   {
      TEDeactivate(editRecHdl);
      DeactivateControl((*docStrucHdl)->vScrollbarHdl);
   }
}

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

void  doOSEvent(EventRecord *eventStrucPtr)
{
   switch((eventStrucPtr->message >> 24) & 0x000000FF)
   {
      case suspendResumeMessage:
         gInBackground = (eventStrucPtr->message & resumeFlag) == 0;
         if(!gInBackground)
            SetCursor(&qd.arrow);
         if(gNumberOfWindows > 0)
            doActivateDocWindow(FrontWindow(),!gInBackground);
         HiliteMenu(0);
         break;

      case mouseMovedMessage:
         doAdjustCursor(FrontWindow());
         break;
   }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doNewDocWindow

WindowPtr  doNewDocWindow(void)
{
   WindowPtr           windowPtr;
   docStructureHandle  docStrucHdl;
   Rect                destAndViewRect;

   if(!(windowPtr = GetNewCWindow(rWindow,NULL,(WindowPtr) -1)))
   {
      doErrorAlert(eWindow);
      return(NULL);
   }

   SetPort(windowPtr);
```

```
    TextSize(10);

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

    gNumberOfWindows ++;

    (*docStrucHdl)->vScrollbarHdl = GetNewControl(rVScrollbar,windowPtr);

    destAndViewRect = windowPtr->portRect;
    destAndViewRect.right -= 15;
    destAndViewRect.bottom -= 15;
    InsetRect(&destAndViewRect,2,2);

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

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

    HUnlock((Handle) docStrucHdl);

    TESetClickLoop(customClickLoopUPP,(*docStrucHdl)->editRecHdl);
    TEAutoView(true,(*docStrucHdl)->editRecHdl);
    TEFeatureFlag(teFOutlineHilite,1,(*docStrucHdl)->editRecHdl);

    return(windowPtr);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ customClickLoop

pascal Boolean  customClickLoop(void)
{
    WindowPtr           windowPtr;
    docStructureHandle  docStrucHdl;
    TEHandle            editRecHdl;
    GrafPtr             oldPort;
    RgnHandle           oldClip;
    Rect                tempRect;
    Point               mouseXY;
    SInt16              linesToScroll = 0;

    windowPtr = FrontWindow();
    docStrucHdl = (docStructureHandle) (GetWRefCon(windowPtr));;
    editRecHdl = (*docStrucHdl)->editRecHdl;

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

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

```
    if(linesToScroll != 0)
        doDrawDataPanel(windowPtr);

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

    return(true);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doSetScrollBarValue

void  doSetScrollBarValue(ControlHandle controlHdl,SInt16 *linesToScroll)
{
    SInt16  controlValue, controlMax;

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

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

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

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

void  doAdjustMenus(void)
{
    MenuHandle              fileMenuHdl, editMenuHdl;
    WindowPtr                   windowPtr;
    docStructureHandle  docStrucHdl;
    TEHandle                editRecHdl;
    SInt32                  scrapOffset;

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

    if(gNumberOfWindows > 0)
    {
        windowPtr = FrontWindow();
        docStrucHdl = (docStructureHandle) (GetWRefCon(windowPtr));;
        editRecHdl = (*docStrucHdl)->editRecHdl;

        EnableItem(fileMenuHdl,iClose);

        if((*editRecHdl)->selStart < (*editRecHdl)->selEnd)
        {
            EnableItem(editMenuHdl,iCut);
            EnableItem(editMenuHdl,iCopy);
            EnableItem(editMenuHdl,iClear);
        }
        else
        {
            DisableItem(editMenuHdl,iCut);
            DisableItem(editMenuHdl,iCopy);
            DisableItem(editMenuHdl,iClear);
        }

        if(GetScrap(NULL,'TEXT',&scrapOffset) > 0)
            EnableItem(editMenuHdl,iPaste);
        else
            DisableItem(editMenuHdl,iPaste);

        if((*editRecHdl)->teLength > 0)
        {
            EnableItem(fileMenuHdl,iSaveAs);
            EnableItem(editMenuHdl,iSelectAll);
        }
        else
        {
            DisableItem(fileMenuHdl,iSaveAs);
            DisableItem(editMenuHdl,iSelectAll);
        }
    }
```

```
  else
  {
    DisableItem(fileMenuHdl,iClose);
    DisableItem(fileMenuHdl,iSaveAs);
    DisableItem(editMenuHdl,iClear);
    DisableItem(editMenuHdl,iSelectAll);
  }

  DrawMenuBar();
}

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

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

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

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mApple:
      if(menuItem == iAbout)
        SysBeep(10);
      else
      {
        GetMenuItemText(GetMenuHandle(mApple),menuItem,itemName);
        daDriverRefNum = OpenDeskAcc(itemName);
      }
      break;

    case mFile:
      doFileMenu(menuItem);
      break;

    case mEdit:
      doEditMenu(menuItem);
      break;

    case kHMHelpMenuID:
      if(FrontWindow())
        doActivateDocWindow(FrontWindow(),false);
      doHelpMenu(menuItem);
      break;
  }

  HiliteMenu(0);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doFileMenu

void  doFileMenu(SInt16 menuItem)
{
  WindowPtr           windowPtr;
  docStructureHandle  docStrucHdl;
  TEHandle            editRecHdl;

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

    case iOpen:
      doOpenCommand();
      break;

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

    case iSaveAs:
```

```
      docStrucHdl = (docStructureHandle) (GetWRefCon(FrontWindow()));
      editRecHdl = (*docStrucHdl)->editRecHdl;
      doSaveAsFile(editRecHdl);
      break;

    case iQuit:
      gDone = true;
      break;
  }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doEditMenu

void  doEditMenu(SInt16 menuItem)
{
  WindowPtr             windowPtr;
  docStructureHandle  docStrucHdl;
  TEHandle            editRecHdl;
  SInt32              totalSize, contigSize, newSize, scrapOffset;
  SInt16              selectionLength;

  windowPtr = FrontWindow();
  docStrucHdl = (docStructureHandle) (GetWRefCon(windowPtr));;
  editRecHdl = (*docStrucHdl)->editRecHdl;

  switch(menuItem)
  {
    case iUndo:
      break;

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

    case iCopy:
      if(ZeroScrap() == noErr)
      {
        TECopy(editRecHdl);
        if(TEToScrap() != noErr)
          ZeroScrap();
      }
      break;

    case iPaste:
      newSize = (*editRecHdl)->teLength + GetScrap(NULL,'TEXT',&scrapOffset);
      if(newSize > kMaxTELength)
        doErrorAlert(eNoSpacePaste);
      else
      {
        if(TEFromScrap() == noErr)
        {
          TEPaste(editRecHdl);
          doAdjustScrollbar(windowPtr);
        }
      }
      break;

    case iClear:
      TEDelete(editRecHdl);
      doAdjustScrollbar(windowPtr);
      break;

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

```
  doDrawDataPanel(windowPtr);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doGetSelectLength

SInt16  doGetSelectLength(TEHandle editRecHdl)
{
  SInt16  selectionLength;

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

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doAdjustScrollbar

void  doAdjustScrollbar(WindowPtr windowPtr)
{
  docStructureHandle docStrucHdl;
  TEHandle           editRecHdl;
  SInt16             numberOfLines, controlMax, controlValue;

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowPtr));;
  editRecHdl = (*docStrucHdl)->editRecHdl;

  numberOfLines = (*editRecHdl)->nLines;
  if(*(*(*editRecHdl)->hText + (*editRecHdl)->teLength - 1) == kReturn)
    numberOfLines += 1;

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

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

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

#if TARGET_CPU_PPC
  if(gMacOS85Present)
  {
    SetControlViewSize((*docStrucHdl)->vScrollbarHdl,(*editRecHdl)->viewRect.bottom -
                       (*editRecHdl)->viewRect.top);
  }
#endif

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

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doAdjustCursor

void  doAdjustCursor(WindowPtr windowPtr)
{
  GrafPtr    oldPort;
  RgnHandle  arrowRegion, iBeamRegion;
  Rect       cursorRect;
  Point      mouseXY;

  if(gInBackground)
  {
    SetCursor(&qd.arrow);
    return;
  }

  GetPort(&oldPort);
  SetPort(windowPtr);

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

```
    cursorRect = windowPtr->portRect;
    cursorRect.bottom -= 15;
    cursorRect.right    -= 15;
    LocalToGlobal(&topLeft(cursorRect));
    LocalToGlobal(&botRight(cursorRect));

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

    GetMouse(&mouseXY);
    LocalToGlobal(&mouseXY);

    if(PtInRgn(mouseXY,iBeamRegion))
    {
        SetCursor(*(GetCursor(iBeamCursor)));
        CopyRgn(iBeamRegion,gCursorRegion);
    }
    else
    {
        SetCursor(&qd.arrow);
        CopyRgn(arrowRegion,gCursorRegion);
    }

    DisposeRgn(arrowRegion);
    DisposeRgn(iBeamRegion);

    SetPort(oldPort);
}

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

void  doCloseWindow(WindowPtr windowPtr)
{
    docStructureHandle  docStrucHdl;

    docStrucHdl = (docStructureHandle) (GetWRefCon(windowPtr));;

    DisposeControl((*docStrucHdl)->vScrollbarHdl);
    TEDispose((*docStrucHdl)->editRecHdl);
    DisposeHandle((Handle) docStrucHdl);
    DisposeWindow(windowPtr);

    gNumberOfWindows --;
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doSaveAsFile

void  doSaveAsFile(TEHandle editRecHdl)
{
    StandardFileReplyfileReply;
    WindowPtr                windowPtr;
    SInt16                fileRefNum;
    SInt32                dataLength;
    Handle                editTextHdl;

    StandardPutFile("\pSave as:","\pUntitled",&fileReply);
    if(fileReply.sfGood)
    {
        windowPtr = FrontWindow();
        SetWTitle(windowPtr,fileReply.sfFile.name);

        if(!(fileReply.sfReplacing))
            FSpCreate(&fileReply.sfFile,' KJB','TEXT',fileReply.sfScript);

        FSpOpenDF(&fileReply.sfFile,fsCurPerm,&fileRefNum);

        dataLength = (*editRecHdl)->teLength;
        editTextHdl = (*editRecHdl)->hText;
        FSWrite(fileRefNum,&dataLength,*editTextHdl);

        FSClose(fileRefNum);
    }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doOpenCommand

void  doOpenCommand(void)
{
```

```
    StandardFileReply fileReply;
    SFTypeList        fileTypes;

    fileTypes[0] = 'TEXT';

    StandardGetFile(NULL,1,fileTypes,&fileReply);
    if(fileReply.sfGood)
      doOpenFile(fileReply.sfFile);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doOpenFile

void  doOpenFile(FSSpec fileSpec)
{
    WindowPtr          windowPtr;
    docStructureHandle docStrucHdl;
    TEHandle           editRecHdl;
    SInt16             fileRefNum;
    SInt32             textLength;
    Handle             textBuffer;

    if((windowPtr = doNewDocWindow()) == NULL)
      return;

    docStrucHdl = (docStructureHandle) (GetWRefCon(windowPtr));;
    editRecHdl = (*docStrucHdl)->editRecHdl;

    SetWTitle(windowPtr,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,editRecHdl);

    HUnlock(textBuffer);
    DisposeHandle(textBuffer);

    FSClose(fileRefNum);

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

    ShowWindow(windowPtr);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doDrawDataPanel

void  doDrawDataPanel(WindowPtr windowPtr)
{
    docStructureHandle docStrucHdl;
    TEHandle           editRecHdl;
    RGBColor           whiteColour = { 0xFFFF, 0xFFFF, 0xFFFF };
    RGBColor           blackColour = { 0x0000, 0x0000, 0x0000 };
    RGBColor           blueColour = { 0x1818, 0x4B4B, 0x8181 };
    ControlHandle      controlHdl;
    Rect               panelRect;
    Str255             textString;

    docStrucHdl = (docStructureHandle) (GetWRefCon(windowPtr));;
    editRecHdl = (*docStrucHdl)->editRecHdl;
    controlHdl = (*docStrucHdl)->vScrollbarHdl;

    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) (*editRecHdl)->teLength,textString);
  MoveTo(47,295);
  DrawString(textString);

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

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

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

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

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

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

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

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

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

  GetIndString(errorString,rErrorStrings,errorCode);

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

```
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doHelpMenu

void  doHelpMenu(SInt16 menuItem)
{
    MenuHandle helpMenuHdl;
    SInt16          origHelpItems, numItems;

    HMGetHelpMenuHandle(&helpMenuHdl);

    numItems = CountMenuItems(helpMenuHdl);
    origHelpItems = numItems - 1;

    if(menuItem > origHelpItems)
        doHelp();
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
// HelpDialog.c
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

#include <Appearance.h>
#include <ControlDefinitions.h>
#include <Dialogs.h>
#include <Resources.h>
#include <TextUtils.h>

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

#define rHelpModal                   128
#define  iUserPane                   2
#define  iScrollBar                  3
#define  iPopupMenu                      4
#define  eHelpDialog                 9
#define  eHelpDocRecord              10
#define  eHelpText                   11
#define  eHelpPicture                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                  editRecHdl;
    ControlHandle           scrollbarHdl;
    SInt16                      pictCount;
    pictInfoStructure   *pictInfoStructurePtr;
}  docStructure, ** docStructureHandle;

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

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

ModalFilterUPP                eventFilterUPP;
ControlUserPaneDrawUPP userPaneDrawFunctionUPP;
ControlActionUPP            actionFunctionUPP;
backColourPattern           gBackColourPattern;
SInt16                            gTextResourceID;
```

```
SInt16                          gPictResourceBaseID;
RgnHandle                          gSavedClipRgn = NULL;

extern Boolean              gMacOS85Present;

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

void                  doHelp                      (void);
void                  doCloseHelp                 (DialogPtr,GrafPtr);
void                  doDisposeDescriptors   (void);
pascal void      userPaneDrawFunction (ControlHandle,SInt16);
Boolean                  doGetText                      (DialogPtr,SInt16,Rect);
Boolean                  doGetPictureInfo          (DialogPtr,SInt16);
pascal void         actionFunction             (ControlHandle,SInt16);
void                  doScrollTextAndPicts    (DialogPtr);
void                  doDrawPictures              (DialogPtr,Rect *);
pascal Boolean eventFilter                (DialogPtr,EventRecord *,SInt16 *);
void                  doSaveBackground          (backColourPattern *);
void                  doRestoreBackground      (backColourPattern *);
void                  doSetBackgroundWhite    (void);

extern void         doUpdate                     (EventRecord *);
extern void         doErrorAlert               (SInt16);

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doHelp

void  doHelp(void)
{
   DialogPtr                   dialogPtr;
   docStructureHandle  docStrucHdl;
   GrafPtr                     oldPort;
   ControlHandle            controlHdl;
   SInt16                      itemHit, menuItem;
   Rect                          userItemRect, destRect, viewRect;

   // .................................................................................................................................... create 
routine descriptors

   eventFilterUPP                = NewModalFilterProc((ProcPtr) eventFilter);
   userPaneDrawFunctionUPP   = NewControlUserPaneDrawProc((ProcPtr) userPaneDrawFunction);
   actionFunctionUPP              = NewControlActionProc((ProcPtr) actionFunction);

   // ........................................................................................................ create dialog and attach document 
structure

   if(!(dialogPtr = GetNewDialog(rHelpModal,NULL,(WindowPtr) -1)))
   {
      doErrorAlert(eHelpDialog);
      doDisposeDescriptors();
      return;
   }

   if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
   {
      doErrorAlert(eHelpDocRecord);
      DisposeDialog(dialogPtr);
      doDisposeDescriptors();
      return;
   }

   SetWRefCon(dialogPtr,(SInt32) docStrucHdl);

   // ....................................................................................................................... set graphics port and 
default button

   GetPort(&oldPort);
   SetPort(dialogPtr);
   SetDialogDefaultItem(dialogPtr,kStdOkItemIndex);

   // ....................................................................................................................................... set user pane 
drawing function

   GetDialogItemAsControl(dialogPtr,iUserPane,&controlHdl);
   SetControlData(controlHdl,kControlNoPart,kControlUserPaneDrawProcTag,
                            sizeof(userPaneDrawFunctionUPP),(Ptr) &userPaneDrawFunctionUPP);

   // ......................................................................... set destination and view rectangles, create edit structure
```

```
userItemRect = (*controlHdl)->contrlRect;
InsetRect(&userItemRect,kTextInset,kTextInset / 2);
destRect = viewRect = userItemRect;
(*docStrucHdl)->editRecHdl = TEStyleNew(&destRect,&viewRect);

// ..................................................... assign handle to scroll bar to relevant document structure field

GetDialogItemAsControl(dialogPtr,iScrollBar,&controlHdl);
(*docStrucHdl)->scrollbarHdl = controlHdl;

// ..................................... 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(dialogPtr,gTextResourceID,viewRect)))
{
   doCloseHelp(dialogPtr,oldPort);
   doDisposeDescriptors();
   return;
}

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

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

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

   gSavedClipRgn = NewRgn();

   // ..................................................................................... show window and save background colour and pattern

   ShowWindow(dialogPtr);

   doSaveBackground(&gBackColourPattern);

   //
.....................................................................................................................................
enter ModalDialog loop

   do
   {
      ModalDialog(eventFilterUPP,&itemHit);

      if(itemHit == iPopupMenu)
      {
         SetControlValue((*docStrucHdl)->scrollbarHdl,0);

         GetDialogItemAsControl(dialogPtr,iPopupMenu,&controlHdl);
         menuItem = GetControlValue(controlHdl);

         switch(menuItem)
         {
            case 1:
               gTextResourceID     = rTextIntroduction;
               gPictResourceBaseID = rPictIntroductionBase;
               break;

            case 2:
               gTextResourceID     = rTextCreatingText;
               gPictResourceBaseID = rPictCreatingTextBase;
               break;

            case 3:
               gTextResourceID     = rTextModifyHelp;
               break;
         }
```

```
            doSetBackgroundWhite();

            if(!(doGetText(dialogPtr,gTextResourceID,viewRect)))
            {
                doCloseHelp(dialogPtr,oldPort);
                doDisposeDescriptors();
                return;
            }
            if(!(doGetPictureInfo(dialogPtr,gPictResourceBaseID)))
            {
                doCloseHelp(dialogPtr,oldPort);
                doDisposeDescriptors();
                return;
            }

            doDrawPictures(dialogPtr,&viewRect);

            doRestoreBackground(&gBackColourPattern);
        }

    } while(itemHit != kStdOkItemIndex);

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

    doCloseHelp(dialogPtr,oldPort);
    doDisposeDescriptors();
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doCloseHelp

void  doCloseHelp(DialogPtr dialogPtr,GrafPtr oldPort)
{
    docStructureHandle  docStrucHdl;
    TEHandle            editRecHdl;
    SInt16              a;

    docStrucHdl = (docStructureHandle) GetWRefCon(dialogPtr);
    editRecHdl = (*docStrucHdl)->editRecHdl;

    if(gSavedClipRgn)
        DisposeRgn(gSavedClipRgn);

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

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

    DisposeHandle((Handle) docStrucHdl);
    DisposeDialog(dialogPtr);

    SetPort(oldPort);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doDisposeDescriptors

void  doDisposeDescriptors(void)
{
    DisposeRoutineDescriptor(eventFilterUPP);
    DisposeRoutineDescriptor(userPaneDrawFunctionUPP);
    DisposeRoutineDescriptor(actionFunctionUPP);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ userPaneDrawFunction

pascal void  userPaneDrawFunction(ControlHandle controlHdl,SInt16 thePart)
{
    Rect                itemRect, viewRect;
    docStructureHandle  docStrucHdl;
    TEHandle            editRecHdl;
    DialogPtr           dialogPtr;
    Boolean             inState;
```

```
    dialogPtr = (*controlHdl)->contrlOwner;
    itemRect = (*controlHdl)->contrlRect;
    InsetRect(&itemRect,1,1);
    itemRect.right += 15;

    DrawThemeListBoxFrame(&itemRect,kThemeStateActive);
    if(((WindowPeek) dialogPtr)->visible)
        inState = ((WindowPeek) dialogPtr)->hilited;
    DrawThemeListBoxFrame(&itemRect,inState);

    doSetBackgroundWhite();
    EraseRect(&itemRect);

    docStrucHdl = (docStructureHandle) GetWRefCon(dialogPtr);
    editRecHdl = (*docStrucHdl)->editRecHdl;
    viewRect = (*editRecHdl)->viewRect;

    TEUpdate(&viewRect,editRecHdl);
    doDrawPictures(dialogPtr,&viewRect);

    doRestoreBackground(&gBackColourPattern);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doGetText

Boolean  doGetText(DialogPtr dialogPtr,SInt16 textResourceID,Rect viewRect)
{
    docStructureHandle  docStrucHdl;
    TEHandle            editRecHdl;
    Handle              helpTextHdl;
    StScrpHandle        stylScrpRecHdl;
    SInt16              numberOfLines, heightOfText, heightToScroll;

    docStrucHdl = (docStructureHandle) GetWRefCon(dialogPtr);
    editRecHdl = (*docStrucHdl)->editRecHdl;

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

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

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

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

    TEStyleInsert(*helpTextHdl,GetHandleSize(helpTextHdl),stylScrpRecHdl,editRecHdl);

    ReleaseResource(helpTextHdl);
    ReleaseResource((Handle) stylScrpRecHdl);

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

    if(heightOfText > (viewRect.bottom - viewRect.top))
    {
        heightToScroll = TEGetHeight((SInt32) numberOfLines,1,editRecHdl) -
                                                (viewRect.bottom - viewRect.top);
        SetControlMaximum((*docStrucHdl)->scrollbarHdl,heightToScroll);
        ActivateControl((*docStrucHdl)->scrollbarHdl);
#if TARGET_CPU_PPC
        if(gMacOS85Present)
        {
            SetControlViewSize((*docStrucHdl)->scrollbarHdl,(*editRecHdl)->viewRect.bottom -
                                    (*editRecHdl)->viewRect.top);
        }
#endif
    }
```

```
    else
    {
        DeactivateControl((*docStrucHdl)->scrollbarHdl);
    }

    return true;
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doGetPictureInfo

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

    docStrucHdl = (docStructureHandle) GetWRefCon(dialogPtr);

    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;

    editRecHdl = (*docStrucHdl)->editRecHdl;
    textHdl = (*editRecHdl)->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,editRecHdl);

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

```
      OffsetRect(&pictInfoPtr[a].bounds,
                 (((*editRecHdl)->destRect.right + (*editRecHdl)->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

pascal void  actionFunction(ControlHandle scrollbarHdl,SInt16 partCode)
{
   docStructureHandle docStrucHdl;
   DialogPtr          dialogPtr;
   TEHandle           editRecHdl;
   SInt16             delta, oldValue, offset, lineHeight, fontAscent;
   Point              thePoint;
   Rect               viewRect;
   TextStyle          style;

   if(partCode)
   {
      dialogPtr = (*scrollbarHdl)->contrlOwner;
      docStrucHdl = (docStructureHandle) GetWRefCon(dialogPtr);
      editRecHdl = (*docStrucHdl)->editRecHdl;
      viewRect = (*editRecHdl)->viewRect;
      thePoint.h = viewRect.left + kTextInset;

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

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

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

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

```
          if(offset == (**editRecHdl).teLength)
            delta += lineHeight;
          break;
      }

      oldValue = GetControlValue(scrollbarHdl);

      if(((delta < 0) && (oldValue > 0)) || ((delta > 0) &&
         (oldValue < GetControlMaximum(scrollbarHdl))))
      {
        GetClip(gSavedClipRgn);
        ClipRect(&dialogPtr->portRect);

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

    doScrollTextAndPicts(dialogPtr);
  }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doScrollTextAndPicts

void  doScrollTextAndPicts(DialogPtr dialogPtr)
{
  docStructureHandle  docStrucHdl;
  TEHandle            editRecHdl;
  SInt16              scrollDistance, oldScroll;
  Rect                updateRect;

  doSetBackgroundWhite();

  docStrucHdl = (docStructureHandle) GetWRefCon(dialogPtr);
  editRecHdl = (*docStrucHdl)->editRecHdl;

  oldScroll = (*editRecHdl)->viewRect.top -(*editRecHdl)->destRect.top;
  scrollDistance = oldScroll - GetControlValue((*docStrucHdl)->scrollbarHdl);
  if(scrollDistance == 0)
  {
    doRestoreBackground(&gBackColourPattern);
    return;

  }
  TEScroll(0,scrollDistance,editRecHdl);

  if((*docStrucHdl)->pictCount == 0)
  {
    doRestoreBackground(&gBackColourPattern);
    return;
  }

  updateRect = (*editRecHdl)->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(dialogPtr,&updateRect);

  doRestoreBackground(&gBackColourPattern);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doDrawPictures

void  doDrawPictures(DialogPtr dialogPtr,Rect *updateRect)
{
  docStructureHandle  docStrucHdl;
  TEHandle            editRecHdl;
  SInt16              pictCount, pictIndex, vOffset;
  PicHandle           thePictHdl;
  Rect                pictLocRect, dummyRect;
```

```
  docStrucHdl = (docStructureHandle) GetWRefCon(dialogPtr);
  editRecHdl = (*docStrucHdl)->editRecHdl;

  vOffset = (*editRecHdl)->destRect.top - (*editRecHdl)->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);
  }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ eventFilter

pascal Boolean  eventFilter(DialogPtr dialogPtr,EventRecord *eventStrucPtr,
                            SInt16 *itemHit)
{
  Boolean            handledEvent;
  GrafPtr            oldPort;
  Point              mouseXY;
  ControlHandle      controlHdl;
  docStructureHandle docStrucHdl;

  handledEvent = false;

  if((eventStrucPtr->what == updateEvt) &&
     ((WindowPtr) eventStrucPtr->message != dialogPtr))
  {
    doUpdate(eventStrucPtr);
  }
  else
  {
    GetPort(&oldPort);
    SetPort(dialogPtr);

    if(eventStrucPtr->what == mouseDown)
    {
      mouseXY = eventStrucPtr->where;
      GlobalToLocal(&mouseXY);
      if(FindControl(mouseXY,dialogPtr,&controlHdl))
      {
        docStrucHdl = (docStructureHandle) GetWRefCon(dialogPtr);
        if(controlHdl == (*docStrucHdl)->scrollbarHdl)
        {
          TrackControl((*docStrucHdl)->scrollbarHdl,mouseXY,actionFunctionUPP);
          *itemHit = iScrollBar;
          handledEvent = true;
        }
      }
    }
    else
    {
      handledEvent = StdFilterProc(dialogPtr,eventStrucPtr,itemHit);
    }

    SetPort(oldPort);
  }

  return(handledEvent);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doSaveBackground
```

```
void  doSaveBackground(backColourPattern *gBackColourPattern)
{
   GrafPtr currentPort;

   GetPort(&currentPort);

   GetBackColor(&gBackColourPattern->backColour);
   gBackColourPattern->backPixelPattern    = NULL;

   if((**((CGrafPtr) currentPort)->bkPixPat).patType != 0)
      gBackColourPattern->backPixelPattern = ((CGrafPtr) currentPort)->bkPixPat;
   else
      gBackColourPattern->backBitPattern =
                                          *(PatPtr) (*(**((CGrafPtr) currentPort)->bkPixPat).patData);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doRestoreBackground

void  doRestoreBackground(backColourPattern *gBackColourPattern)
{
   RGBBackColor(&gBackColourPattern->backColour);

   if(gBackColourPattern->backPixelPattern)
      BackPixPat(gBackColourPattern->backPixelPattern);
   else
      BackPat(&gBackColourPattern->backBitPattern);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doSetBackgroundWhite

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

   RGBBackColor(&whiteColour);
   BackPat(&qd.white);
}

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

Demonstration Program Text1 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.
- 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 (live scrolling is used), clicking once in a scroll arrow or gray area, and holding the mouse down in a scroll arrow or gray area.

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 Text1 Help from the 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.

Text1.c

#define

kMaxTELength represents the maximum allowable number of bytes in a TextEdit edit structure. kTab, kDel, and kReturn representing the character codes generated by the tab, delete and return keys.

#typedef

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

Global Variables

scrollActionFunctionUPP will be assigned a universal procedure pointer to an action function for the scroll bar. customClickLoopUPP will be assigned a universal procedure pointer to a custom click loop function. gMacOS85Present will be assigned true if Mac OS 8.5 or later is present. gCursorRegion is related to the WaitNextEvent's mouseRgn parameter. gNumberOfWindows will keep track of the number of windows open at any one time. gOldControlValue will be assigned the scroll bar's control value.

main

The main function initialises the system software managers, sets up the menus, opens a new document window and enters the event loop. gMacOS85Present is assigned true of Mac OS 8.5 or later is present. This global variable will govern whether certain Control Manager functions relating to proportional scroll boxes, and which are available only in Mac OS 8.5 or later, are called.

eventLoop

Note that WaitNextEvent's sleep parameter is assigned the value returned by the call to LMGetCaretTime.

If WaitNextEvent returns a null event, and provided at least one window is open, the application-defined function doIdle is called.

doIdle

doIdle is called whenever a NULL event is received.

The first line gets a pointer to the front window, allowing the next line to attempt to retrieve a handle to that window's document structure. If the attempt is successful, TEIdle is called to blink the insertion point.

doEvents

Note that, in the case of a mouse-down event in the content area, the application-defined function doInContent is called and that, in the case of key events, the application-defined function doKeyEvent is called provided the key press is not a Command key equivalent.

doKeyEvent

doKeyEvent handles all key-down events that are not Command key equivalents.

The first three lines get a handle to the edit structure which forms part of 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 edit structure. If this is zero (that is, there is no selection range and an insertion point is being displayed), the selEnd field is increased by one. This, in effect, creates a selection range comprising the character following the insertion point. TEDelete deletes the current selection range from the edit structure. Such deletions could change the number of text lines in the edit structure, requiring the vertical scroll bar to be adjusted; hence the call to the application-defined 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 in the edit structure 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 length of the edit structure minus the selection length plus 1 is less than 32,767, the character code is passed to TEKey for insertion into the edit structure. In addition, and since all this could change the number of lines in the edit structure, the scroll bar adjustment function is called.

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

The last line calls the application-defined 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 hook function which will be repeatedly called by TrackControl while the mouse button remains down in the scroll box, scroll arrows or gray areas of the vertical scroll bar.

The first line gets a pointer to the window which "owns" the control. The next two lines get a handle to the edit structure associated with the window.

Within the outer if block, the first if block executes if the control part is not the scroll box (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, 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 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.

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 edit 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 (indicator), the variable linesToScroll is assigned a value equal to the control's value as it was 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 application-defined function which prints data extracted from the edit and control structures at the bottom of the window.

doInContent

doInContent continues mouse-down processing.

The first three lines retrieve a handle to the edit structure associated with that window.

The next two lines convert the mouse-down coordinates from global to local coordinates. (Local coordinates will be required by upcoming calls to FindControl, TrackControl, and PtInRect.)

If the mouse-down was in a control (that is, the scroll bar), the global variable gOldControlValue is assigned the value of the control to cater for the case of the mouse-down being within the scroll box (indicator), and TrackControl is called with the universal procedure pointer to the application defined action function passed in the third parameter. TrackControl retains control until the mouse button is released, during which time the previously described action function scrollActionFunction is repeatedly called.

If the mouse-down was not in a control, PtInRect checks whether it 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, a check is made of the shift key position at the time of the mouse-down. The result is passed as the second parameter in the call to TEClick. (TEClick's behaviour depends on the position of the shift key.)

doUpdate

doUpdate handles update events. The first three lines get the handle to the edit structure associated with the window.

Between the usual BeginUpdate and EndUpdate calls, TEUpdate is called to draw the text in the edit structure, UpdateControls is called to draw the scroll bar, and the data panel is redrawn.

doActivateDocWindow

doActivateDocWindow handles window activation/deactivation.

The first two lines retrieve a handle to the edit 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 edit 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 edit structure associated with the window, ActivateControl activates the scroll bar, and doAdjustScrollbar adjusts the scroll bar.

If the window is becoming inactive, TEDeactivate deactivates the edit 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, attaches a document structure to that window, creates a vertical scroll bar, creates a monostyled edit structure, installs the custom click loop function, enables automatic scrolling, and enables outline highlighting.

GetNewCWindow opens a new window and sets its colour graphics port as the current colour graphics port. (Since the edit structure assumes the drawing environment specified in the colour graphics port structure, setting the colour graphics port must be done before the call to TENew to create the edit structure.)

The next two lines set the text size and font. (These will be copied from the colour graphics port to the edit structure when TENew is called.)

The next block creates a document structure and assigns a handle to that structure to the window structure's refCon field. 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 edit 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 function customClickLoop in the clickLoop field of the edit structure. TEAutoView enables automatic scrolling for the edit structure. TEFeatureFlag enables outline highlighting for the edit structure.

The last line returns a pointer to the newly opened window.

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 edit structure associated with the window. The next two lines save the current colour graphics port and set the window's colour 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 edit 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 colour 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, Lines The first three lines in the if block get a handle to the edit structure associated with the front window and the first call to EnableItem 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 any text in the desk scrap (the call to GetScrap), the Paste item is enabled, otherwise it is disabled. If there is any text in the edit 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.

doFileMenu

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

(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 edit structures, TECut, TECopy, and TEPaste do not copy/paste text to/from the desk scrap. This program, however, supports copying/pasting to/from the desk scrap.

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

The iCut case handles the Cut command. Firstly, the call to ZeroScrap attempts to empty the desk 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 edit 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 desk scrap. If the latter call is not successful, ZeroScrap cleans up as best it can by emptying the desk scrap.

The iCopy case handles the Copy command. If the call to empty the desk scrap is successful, TECopy is called to copy the selected text from the edit structure to the TextEdit private scrap. TEToScrap then copies the private scrap to the desk 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 first line establishes a value equal to the number of bytes in the edit structure plus the number of bytes of the specified data type ('TEXT') in the desk scrap. If this value exceeds the TextEdit limit, the user is advised via an error message. Otherwise, TEFromScrap copies the desk scrap to TextEdit's private scrap, TEPaste inserts the private scrap into the edit structure, and the following line adjusts the scroll bar.

The iClear case handles the Clear command. TEDelete deletes the current selection range from the edit 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 edit structure associated with the window in question.

At the next block, the value in the nLines field of the edit 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 edit 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 edit structure less the number of lines which will fit in the view rectangle. If this value is less than 0 (indicating that the number of lines in the edit 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 edit structure contains the value 13, the control value will be set to 3.

If the target is the PowerPC target, and if Mac OS 8.5 or later is present, SetControlViewSize is called to advise the Control Manager of the height of the view rectangle. If Smart Scrolling is selected on in the Options tab of the Mac OS 8.5 and later Appearance control panel, this will cause the scroll boxes of the scroll bar to be proportional.

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. 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 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 at Chapter 13 — Offscreen Graphics Worlds, Pictures, Cursors, and Icons.

doCloseWindow

doCloseWindow disposes of the specified window. The associated scroll bar, the associated edit 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.

> For a complete example of opening and saving monostyled 'TEXT' documents, see the demonstration program at Chapter 16 — Files.

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 edit structure and the contrlValue and contrlMax fields of the scroll bar's control structure.

doHelpMenu

doHelpMenu handles a choice of the Text1 Help item added by this program to the system-managed Help Menu. The code reflects the fact that Apple reserves the right to add items to the Help menu in future versions of the system software.

HMGetHelpMenuHandle gets a handle to the Help menu structure. CountMenuItems returns the number of items in the Help menu. Since we know that we have added one item to this menu, the next line will establish the original number of help items. If the value passed to the doHelpMenu function is greater than this, it must therefore represent the item number of our Text1 Help item, in which case the application-defined function which opens the help dialog is called.

HelpDialog.c

#define

Constants are established for the 'DLOG' resource ID and items in the dialog, 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 the 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 Text1 Help item in the Help menu.

The dialog will utilise an event filter function, a user pane drawing function, and an action function. The first three lines create the associated routine descriptors.

GetNewDialog creates the dialog. NewHandle creates a block for a document structure and the handle is assigned to the dialog's window structure's refCon field.

SetDialogDefaultItem sets the dialog's push button as the default item and ensures that the default ring will be drawn around that item.

At the next block, SetControlData sets the user pane drawing function.

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 edit structure based on those two rectangles.

The next block assigns the handle to the scrollbar to the appropriate field of the dialog's document structure.

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 application-defined function doGetText which, amongst other things, loads the specified 'TEXT'/'styl' resources and inserts the text and style information into the edit structure.

The next block calls the application-defined 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 colour graphic's port's clipping region.

To complete the initial setting up, ShowWindow is called to makes the dialog visible and the application-defined function doSaveBackground is called to save the background colour and pattern of the dialog as they were when the dialog was created.

The do-while ModalDialog loop continues until the user clicks the "Done" (OK) button or hits the Return key.

The modal dialog uses an application-defined filter function which, as will be seen, handles mouse-down events within the scrollbar. The only other event of interest is a hit on the pop-up menu button. Accordingly, if ModalDialog returns a hit on that control, SetControlValue is called to set the scroll bar's value to 0, 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.

Before anything is drawn in that part of the dialog used to display the text and pictures, the background colour and pattern are set to, respectively, the white colour and the white pattern by a call to doSetBackgroundWhite. The next block then 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. With all drawing completed, the application-defined function doRestoreBackground is called to restore the saved background colour and pattern.

When the user clicks the "Done" (OK) button or hits the Return key, application-defined functions are called to close down the Help dialog and dispose of the routine descriptors.

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 region used to save the clipping region. TEDispose disposes of the edit structure. The next block disposes of any 'PICT' resources currently in memory, together with the picture information structure. Finally, the dialog's document structure is disposed of, the dialog itself is disposed of, and the graphics port saved in doHelp is restored.

userPaneDrawFunction

userPaneDrawFunction is the user pane drawing function set within doHelp. It will be called automatically whenever the dialog receives an update event.

The first block gets a pointer to the dialog to which the user pane control belongs, gets the user pane's rectangle, insets that rectangle by one pixel all round, and then further expands it to the right 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 window is the movable modal dialog is currently

the active window. (The first call to DrawThemeListBoxFrame is required so as to ensure that the list box frame is drawn when the dialog is initially displayed.)

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

The three lines retrieve the view rectangle from the edit structure. The call to TEUpdate draws the text in the edit structure in the view rectangle. The call to drawPictures draws any pictures which might currently be located in the view rectangle.

The last line restores the background colour and pattern saved in doHelp.

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 edit structure.

The first two lines get a handle to the edit 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 edit 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 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.

If the target is the PowerPC target, and if Mac OS 8.5 or later is present, SetControlViewSize is called to advise the Control Manager of the height of the view rectangle. If Smart Scrolling is selected on in the Options tab of the Mac OS 8.5 and later Appearance control panel, this will cause the scroll boxes of the scroll bar to be proportional.

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 edit structure, then a handle to the block containing the actual text. This latter is then used at 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 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 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 of the scroll bar, and then call a separate application-defined 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.)

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 edit 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, 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, 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 application-defined 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 application-defined 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 edit 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. (Of course, if the scroll box has been dragged by the user over a large distance, the "vacated" area will equate to the whole view rectangle.) 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 edit 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.

eventFilter

eventFilter is the application-defined filter function for the dialog. It is essentially the same as the event filter introduced in the demonstration program Dialogs (Chapter 8) except that it also intercepts mouse-down events.

In the case of a mouse-down event, the location of the event is determined in local coordinates. If FindControl and the following two lines determine that the mouse-down was within the scrollbar, TrackControl is called to handle user interaction while the mouse button remains down. While the mouse button remains down, TrackControl continually calls actionFunction.

doSaveBackground, doRestoreBackground, and doSetBackgroundWhite

doSaveBackground and doRestoreBackground are essentially the same as the functions introduced at Chapter 12 — Drawing With QuickDraw for saving and restoring the drawing environment. In this case, the saving and restoring is limited to the background colour and the background bit or pixel pattern. doSetBackgroundWhite sets the background colour to white and the background pattern to the pattern white.

Demonstration Program Text2

```
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
// Text2.c
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
//
// This program demonstrates:
//
// •   The use of edit text field controls and TextEdit to enter data into the various
//     fields of a database record.
//
// •   The formatting and display of dates, times and numbers.
//
// The program opens a document window in which four edit text fields are displayed.  The
// first field requires text to be entered, the second an integer value, the third a
// floating point value (that is, a currency value), and the fourth a date.  The user may
// select a field for data entry using the mouse or by cycling between fields using the
// Tab key.
//
// TENew could have been used to create the four edit text fields; however, edit text
// field controls have been used so as to simplify the appearance aspects of window
// activation and deactivation and the setting and erasing of the keyboard focus frame.
// That said, TextEdit functions are widely used to deal directly with the controls' edit
// text structures.  Amongst other things, this allows the program to limit the number of
// characters that may be entered in each field and to cause the forward-delete key to be
// used to forward-delete characters.  (The forward-delete key is not supported by the
// edit text field control CDEF.)
//
// The current date and the current time are displayed at the top of the window.
//
// The program utilises the following resources:
//
// •   An 'MBAR' resource, and 'MENU' resources for Apple, File, and Edit menus (preload,
//     non-purgeable).
//
// •   'CNTL' resources (purgeable) for edit text field controls and primary and secondary
//     group boxes (text title variant).
//
// •   A 'WIND' resource (purgeable) (initially not visible).
//
// •   'hrct' and 'hwin' resources (purgeable) for balloon help.
//
// •   A 'SIZE' resource with the acceptSuspendResumeEvents, doesActivateOnFGSwitch,
//     canBackground and  is32BitCompatible flags set.
//
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
```

```
//
.....
..... includes

#include <Appearance.h>
#include <ControlDefinitions.h>
#include <Devices.h>
#include <fp.h>
#include <LowMem.h>
#include <Sound.h>
#include <ToolUtils.h>

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

#define rMenubar                    128
#define mApple                        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  iSelectAll                 7
#define rWindow                       128
#define cDateAndTimeGroupBox 128
#define cDataEntryGroupBox      129
#define cItemTitleEditText      130
#define cQuantityEditText       131
#define cUnitValueEditText      132
#define cDateEditText           133
#define cLastRecordGroupBox     134
#define cBalloonGroupBox        135
#define kTab                          0x09
#define kDel                        0x7F
#define kReturn                       0x0D
#define kMaxCharasItem          20
#define kMaxCharasQuant               9
#define kMaxCharasValue               9
#define kMaxCharasDate          16
#define topLeft(r)                  (((Point *) &(r))[0])
#define botRight(r)                   (((Point *) &(r))[1])

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

typedef struct
{
   ControlHandle  itemControlHdl;
   ControlHandle  quantControlHdl;
   ControlHandle  valueControlHdl;
   ControlHandle  dateControlHdl;
} docStruc, **docStrucHandle;

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

Boolean       gIsColourDevice;
Boolean       gPixelDepth;
Boolean       gDone;
Boolean       gInBackground;
RgnHandle     gCursorRegion;
TEHandle  gCurrentEditStrucHdl;
SInt16        gMaxCharasThisField;
Rect          gItemRect    = { 89 ,91,102, 231 };
Rect          gQuantRect  = { 112 ,91,125,233 };
Rect          gValueRect   = { 135,91,148,233 };
Rect          gDateRect       = { 158,91,171,233 };
Str255        gLastRecordItem;
Str255        gLastRecordQuantity;
Str255        gLastRecordUnitValue;
Str255        gLastRecordTotalValue;
Str255        gLastRecordReviewDate;
```

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

void      main                     (void);
void      doInitManagers           (void);
void      doGetControls            (WindowPtr);
void      eventLoop                (void);
void      doIdle                   (void);
void      doEvents                 (EventRecord *);
void      doUpdate                 (EventRecord *);
void      doActivate               (EventRecord *);
void      doActivateWindow         (WindowPtr,Boolean);
void      doMenuChoice             (SInt32);
void      doAdjustMenus            (void);
void      doKeyEvent               (SInt8,docStrucHandle);
void      doHandleTabKey           (void);
void      doHandleDelKey           (void);
void      doInContent              (EventRecord *);
void      doChangeCurrentEditRec   (ControlHandle,Boolean,Point,SInt16);
void      doTodaysDate             (void);
void      doAcceptNewRecord        (void);
void      doAcceptValueField       (TEHandle,TEHandle);
void      doAcceptDateField        (TEHandle);
TEHandle  doGetEditTextHandle      (ControlHandle);
void      doDrawWindow             (void);
void      doAdjustCursor           (WindowPtr);
void      doCopyPString            (Str255,Str255);
void      doGetDepthAndDevice      (void);

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

void  main(void)
{
  Handle          menubarHdl;
  MenuHandle      menuHdl;
  WindowPtr       windowPtr;
  docStrucHandle  docStrucHdl;

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

  doInitManagers();

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

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

  menuHdl = GetMenuHandle(mApple);
  if(menuHdl == NULL)
    ExitToShell();
  else
    AppendResMenu(menuHdl,'DRVR');

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

  if(!(windowPtr = GetNewCWindow(rWindow,NULL,(WindowPtr) -1)))
    ExitToShell();

  SetPort(windowPtr);
  TextSize(10);

  SetThemeWindowBackground(windowPtr,kThemeBrushDialogBackgroundActive,true);

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

  if(!(docStrucHdl = (docStrucHandle) NewHandle(sizeof(docStruc))))
    ExitToShell();
  SetWRefCon(windowPtr,(SInt32) docStrucHdl);

  // ................................................................. get controls, show window, save background colour and pattern

  doGetControls(windowPtr);
```

```
    ShowWindow(windowPtr);

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

    doGetDepthAndDevice();

    // 
…………………………………………………………………………………………………………………………………………
…………… enter eventLoop

    eventLoop();
}

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

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

    InitGraf(&qd.thePort);
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(NULL);
    InitCursor();

    FlushEvents(everyEvent,0);

    RegisterAppearanceClient();
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doGetControls

void  doGetControls(WindowPtr windowPtr)
{
    ControlHandle              controlHdl,lastRecordControlHdl;
    docStrucHandle             docStrucHdl;
    Boolean                        booleanData = true;
    ControlFontStyleRec controlFontStyleStruc;

    CreateRootControl(windowPtr,&controlHdl);

    docStrucHdl = (docStrucHandle) (GetWRefCon(windowPtr));

    if(!(controlHdl   = GetNewControl(cDateAndTimeGroupBox,windowPtr)))
        ExitToShell();
    if(!(controlHdl = GetNewControl(cDataEntryGroupBox,windowPtr)))
        ExitToShell();
    if(!((*docStrucHdl)->itemControlHdl = GetNewControl(cItemTitleEditText,windowPtr)))
        ExitToShell();
    if(!((*docStrucHdl)->quantControlHdl = GetNewControl(cQuantityEditText,windowPtr)))
        ExitToShell();
    if(!((*docStrucHdl)->valueControlHdl = GetNewControl(cUnitValueEditText,windowPtr)))
        ExitToShell();
    if(!((*docStrucHdl)->dateControlHdl = GetNewControl(cDateEditText,windowPtr)))
        ExitToShell();
    if(!(lastRecordControlHdl  = GetNewControl(cLastRecordGroupBox,windowPtr)))
        ExitToShell();
    if(!(controlHdl   = GetNewControl(cBalloonGroupBox,windowPtr)))
        ExitToShell();

    controlFontStyleStruc.flags = kControlUseFontMask;
    controlFontStyleStruc.font = kControlFontSmallSystemFont;
    SetControlFontStyle((*docStrucHdl)->itemControlHdl,&controlFontStyleStruc);
    SetControlFontStyle((*docStrucHdl)->quantControlHdl,&controlFontStyleStruc);
    SetControlFontStyle((*docStrucHdl)->valueControlHdl,&controlFontStyleStruc);
    SetControlFontStyle((*docStrucHdl)->dateControlHdl,&controlFontStyleStruc);
    controlFontStyleStruc.font = kControlFontSmallBoldSystemFont;
    SetControlFontStyle(lastRecordControlHdl,&controlFontStyleStruc);

    SetKeyboardFocus(windowPtr,(*docStrucHdl)->itemControlHdl,kControlEditTextPart);
    gCurrentEditStrucHdl = doGetEditTextHandle((*docStrucHdl)->itemControlHdl);
    gMaxCharasThisField = kMaxCharasItem;
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ eventLoop
```

```
void  eventLoop(void)
{
   EventRecord eventStructure;
   Boolean         gotEvent;
   SInt32        sleepTime;

   gDone = false;
   gCursorRegion = NewRgn();
   sleepTime = LMGetCaretTime();

   while(!gDone)
   {
      gotEvent = WaitNextEvent(everyEvent,&eventStructure,sleepTime,gCursorRegion);

      if(gotEvent)
         doEvents(&eventStructure);
      else
         doIdle();
   }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doIdle

void  doIdle(void)
{
   static UInt32    oldRawSeconds;
   UInt32              rawSeconds;
   Str255              timeString;
   Rect                 eraseRect;

   IdleControls(FrontWindow());

   GetDateTime(&rawSeconds);

   if(rawSeconds > oldRawSeconds)
   {
      TimeString(rawSeconds,true,timeString,NULL);
      MoveTo(268,39);
      SetRect(&eraseRect,268,29,323,39);
      EraseRect(&eraseRect);
      DrawString(timeString);
      oldRawSeconds = rawSeconds;
   }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doEvent

void  doEvents(EventRecord *eventStrucPtr)
{
   WindowPtr              windowPtr;
   docStrucHandle   docStrucHdl;
   SInt16              partCode;
   SInt8                charCode;

   windowPtr = FrontWindow();
   docStrucHdl = (docStrucHandle) GetWRefCon(windowPtr);

   switch(eventStrucPtr->what)
   {
      case mouseDown:
         partCode = FindWindow(eventStrucPtr->where,&windowPtr);

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

            case inContent:
               if(windowPtr != FrontWindow())
                  SelectWindow(windowPtr);
               else
                  doInContent(eventStrucPtr);
               break;

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

```
          doAdjustCursor(windowPtr);
          break;

        case inGoAway:
          if(TrackGoAway(windowPtr,eventStrucPtr->where))
            gDone = true;
          break;
      }
      break;

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

    case updateEvt:
      doUpdate(eventStrucPtr);
      break;

    case activateEvt:
      doActivate(eventStrucPtr);
      break;

    case osEvt:
      switch((eventStrucPtr->message >> 24) & 0x000000FF)
      {
        case suspendResumeMessage:
          gInBackground = (eventStrucPtr->message & resumeFlag) == 0;
          if(!gInBackground)
            SetCursor(&qd.arrow);
          doActivateWindow(FrontWindow(),!gInBackground);
          break;

        case mouseMovedMessage:
          doAdjustCursor(FrontWindow());
          break;
      }
      break;
  }
}

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

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

  windowPtr = (WindowPtr) eventStrucPtr->message;
  docStrucHdl = (docStrucHandle) GetWRefCon(windowPtr);

  GetPort(&oldPort);
  SetPort(windowPtr);

  BeginUpdate((WindowPtr) eventStrucPtr->message);
  UpdateControls(windowPtr,windowPtr->visRgn);
  doDrawWindow();
  EndUpdate((WindowPtr) eventStrucPtr->message);

  SetPort(oldPort);
}

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

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

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

```
  doActivateWindow(windowPtr,becomingActive);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doActivateWindow

void  doActivateWindow(WindowPtr windowPtr,Boolean becomingActive)
{
  ControlHandle controlHdl;

  GetRootControl(windowPtr,&controlHdl);

  if(becomingActive)
  {
    ActivateControl(controlHdl);
    doDrawWindow();
  }
  else
  {
    DeactivateControl(controlHdl);
    doDrawWindow();
  }
}

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

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

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

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mApple:
      if(menuItem == iAbout)
        SysBeep(10);
      else
      {
        GetMenuItemText(GetMenuHandle(mApple),menuItem,itemName);
        daDriverRefNum = OpenDeskAcc(itemName);
      }
      break;

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

    case mEdit:
      switch(menuItem)
      {
        case iCut:
          TECut(gCurrentEditStrucHdl);
          break;

        case iCopy:
          TECopy(gCurrentEditStrucHdl);
          break;

        case iPaste:
          TEPaste(gCurrentEditStrucHdl);
          break;

        case iClear:
          TEDelete(gCurrentEditStrucHdl);
          break;

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

```
    HiliteMenu(0);
}

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

void  doAdjustMenus(void)
{
    MenuHandle fileMenuHdl, editMenuHdl;

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

    if((*gCurrentEditStrucHdl)->selStart < (*gCurrentEditStrucHdl)->selEnd)
    {
        EnableItem(editMenuHdl,iCut);
        EnableItem(editMenuHdl,iCopy);
        EnableItem(editMenuHdl,iClear);
    }
    else
    {
        DisableItem(editMenuHdl,iCut);
        DisableItem(editMenuHdl,iCopy);
        DisableItem(editMenuHdl,iClear);
    }

    if(TEGetScrapLength() > 0)
        EnableItem(editMenuHdl,iPaste);
    else
        DisableItem(editMenuHdl,iPaste);

    if((*gCurrentEditStrucHdl)->teLength > 0)
        EnableItem(editMenuHdl,iSelectAll);
    else
        DisableItem(editMenuHdl,iSelectAll);

    DrawMenuBar();
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doKeyEvent

void  doKeyEvent(SInt8 charCode,docStrucHandle docStrucHdl)
{
    if(charCode == kTab)
        doHandleTabKey();
    else if(charCode == kDel)
        doHandleDelKey();
    else if(charCode == kReturn)
        doAcceptNewRecord();
    else
    {
        if(gCurrentEditStrucHdl == doGetEditTextHandle((*docStrucHdl)->quantControlHdl) ||
             gCurrentEditStrucHdl == doGetEditTextHandle((*docStrucHdl)->valueControlHdl))
        {
            if(!((charCode >= '0' && charCode <= '9') || charCode == '.' ||
                        charCode == '-' ||(charCode < 0x20)))
            {
                SysBeep(10);
                return;
            }
        }

        if(((*gCurrentEditStrucHdl)->teLength < gMaxCharasThisField) || (charCode < 0x20))
            TEKey(charCode,gCurrentEditStrucHdl);
        else
            SysBeep(10);
    }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doHandleTabKey

void  doHandleTabKey(void)
{
    WindowPtr          windowPtr;
    docStrucHandle     docStrucHdl;
    ControlHandle      oldControlHdl;
    Point              dummy;

    windowPtr = FrontWindow();
```

```
  docStrucHdl = (docStrucHandle) (GetWRefCon(windowPtr));

  GetKeyboardFocus(windowPtr,&oldControlHdl);

  if(gCurrentEditStrucHdl == doGetEditTextHandle((*docStrucHdl)->itemControlHdl))
    doChangeCurrentEditRec((*docStrucHdl)->quantControlHdl,false,dummy,kMaxCharasQuant);
  else if(gCurrentEditStrucHdl == doGetEditTextHandle((*docStrucHdl)->quantControlHdl))
    doChangeCurrentEditRec((*docStrucHdl)->valueControlHdl,false,dummy,kMaxCharasValue);
  else if(gCurrentEditStrucHdl == doGetEditTextHandle((*docStrucHdl)->valueControlHdl))
    doChangeCurrentEditRec((*docStrucHdl)->dateControlHdl,false,dummy,kMaxCharasDate);
  else if(gCurrentEditStrucHdl == doGetEditTextHandle((*docStrucHdl)->dateControlHdl))
    doChangeCurrentEditRec((*docStrucHdl)->itemControlHdl,false,dummy,kMaxCharasItem);

  Draw1Control(oldControlHdl);

  doAdjustCursor(windowPtr);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doHandleDelKey

void  doHandleDelKey(void)
{
  SInt16  selectionLength;

  selectionLength = (*gCurrentEditStrucHdl)->selEnd - (*gCurrentEditStrucHdl)->selStart;
  if(selectionLength == 0)
    (*gCurrentEditStrucHdl)->selEnd += 1;
  TEDelete(gCurrentEditStrucHdl);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doInContent

void  doInContent(EventRecord *eventStrucPtr)
{
  WindowPtr        windowPtr;
  docStrucHandle   docStrucHdl;
  Boolean          shiftKeyPosition;
  ControlHandle  oldControlHdl, controlHdl;

  windowPtr = FrontWindow();
  docStrucHdl = (docStrucHandle) GetWRefCon(windowPtr);
  GetKeyboardFocus(windowPtr,&oldControlHdl);
  shiftKeyPosition = (eventStrucPtr->modifiers & shiftKey) != 0;

  SetPort(windowPtr);
  GlobalToLocal(&eventStrucPtr->where);

  if(FindControl(eventStrucPtr->where,windowPtr,&controlHdl))
  {
    if(controlHdl == (*docStrucHdl)->itemControlHdl)
    {
      if(gCurrentEditStrucHdl == doGetEditTextHandle((*docStrucHdl)->itemControlHdl))
        TEClick(eventStrucPtr->where,shiftKeyPosition,
                doGetEditTextHandle((*docStrucHdl)->itemControlHdl));
      else
      {
        doChangeCurrentEditRec((*docStrucHdl)->itemControlHdl,true,eventStrucPtr->where,
                               kMaxCharasItem);
      }
    }
    else if(controlHdl == (*docStrucHdl)->quantControlHdl)
    {
      if(gCurrentEditStrucHdl == doGetEditTextHandle((*docStrucHdl)->quantControlHdl))
        TEClick(eventStrucPtr->where,shiftKeyPosition,
                doGetEditTextHandle((*docStrucHdl)->quantControlHdl));
      else
        doChangeCurrentEditRec((*docStrucHdl)->quantControlHdl,true,eventStrucPtr->where,
                               kMaxCharasQuant);
    }
    else if(controlHdl == (*docStrucHdl)->valueControlHdl)
    {
      if(gCurrentEditStrucHdl == doGetEditTextHandle((*docStrucHdl)->valueControlHdl))
        TEClick(eventStrucPtr->where,shiftKeyPosition,
                doGetEditTextHandle((*docStrucHdl)->valueControlHdl));
      else
        doChangeCurrentEditRec((*docStrucHdl)->valueControlHdl,true,eventStrucPtr->where,
                               kMaxCharasValue);
    }
    else if(controlHdl == (*docStrucHdl)->dateControlHdl)
```

```
        {
            if(gCurrentEditStrucHdl == doGetEditTextHandle((*docStrucHdl)->dateControlHdl))
                TEClick(eventStrucPtr->where,shiftKeyPosition,
                            doGetEditTextHandle((*docStrucHdl)->dateControlHdl));
            else
                doChangeCurrentEditRec((*docStrucHdl)->dateControlHdl,true,eventStrucPtr->where,
                                                    kMaxCharasDate);
        }
    }

    Draw1Control(oldControlHdl);
    doAdjustCursor(windowPtr);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doChangeCurrentEditRec

void  doChangeCurrentEditRec(ControlHandle newControlHdl,Boolean teClickFlag,
                                                Point mouseXY,SInt16 maxCharas)
{
    ClearKeyboardFocus(FrontWindow());
    SetKeyboardFocus(FrontWindow(),newControlHdl,kControlEditTextPart);

    gCurrentEditStrucHdl = doGetEditTextHandle(newControlHdl);

    if(teClickFlag)
        TEClick(mouseXY,false,doGetEditTextHandle(newControlHdl));

    TESetSelect(0,32767,gCurrentEditStrucHdl);

    gMaxCharasThisField = maxCharas;
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doTodaysDate

void  doTodaysDate(void)
{
    UInt32  rawSeconds;
    Str255  dateString;

    GetDateTime(&rawSeconds);
    DateString(rawSeconds,longDate,dateString,NULL);
    MoveTo(59,39);
    DrawString(dateString);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doAcceptNewRecord

void  doAcceptNewRecord(void)
{
    WindowPtr           windowPtr;
    docStrucHandle   docStrucHdl;
    TEHandle            itemEditHdl, quantEditHdl, valueEditHdl, dateEditHdl;
    Rect                theRect;
    Point               dummy;

    windowPtr = FrontWindow();
    docStrucHdl = (docStrucHandle) GetWRefCon(windowPtr);

    itemEditHdl     = doGetEditTextHandle((*docStrucHdl)->itemControlHdl);
    quantEditHdl    = doGetEditTextHandle((*docStrucHdl)->quantControlHdl);
    valueEditHdl = doGetEditTextHandle((*docStrucHdl)->valueControlHdl);
    dateEditHdl     = doGetEditTextHandle((*docStrucHdl)->dateControlHdl);

    if((*itemEditHdl)->teLength   == 0 || (*quantEditHdl)->teLength == 0 ||
        (*valueEditHdl)->teLength == 0 || (*dateEditHdl)->teLength   == 0)
    {
        SysBeep(10);
        return;
    }

    GetDialogItemText((*itemEditHdl)->hText,gLastRecordItem);
    GetDialogItemText((*quantEditHdl)->hText,gLastRecordQuantity);
    doAcceptValueField(valueEditHdl,quantEditHdl);
    doAcceptDateField(dateEditHdl);

    SetRect(&theRect,144,198,310,300);
    InvalRect(&theRect);

    TESetSelect(0,32767,itemEditHdl);
```

```
    TEDelete(itemEditHdl);

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

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

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

    gCurrentEditStrucHdl = itemEditHdl;
    doChangeCurrentEditRec((*docStrucHdl)->itemControlHdl,false,dummy,kMaxCharasItem);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doAcceptValueField

void  doAcceptValueField(TEHandle valueEditHdl,TEHandle quantEditHdl)
{
    Handle                itl4ResourceHdl;
    SInt32                numpartsOffset;
    SInt32                numpartsLength;
    NumberParts              *numpartsTablePtr;
    Str255                formatString = "\p'$'###,###,###.00;'Valueless';'Valueless'";
    NumFormatString       formatStringRec;
    Str255                inputNumString;
    Str255                formattedNumString;
    extended80            value80Bit;
    Str255                quantityString;
    SInt32                quantity;
    FormatResultType      result;
#if TARGET_CPU_PPC
    double                valueDouble;
#endif

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

    StringToFormatRec(formatString,numpartsTablePtr,&formatStringRec);

    GetDialogItemText((*valueEditHdl)->hText,inputNumString);

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

    doCopyPString(formattedNumString,gLastRecordUnitValue);

    GetDialogItemText((*quantEditHdl)->hText,quantityString);
    StringToNum(quantityString,&quantity);

#if TARGET_CPU_68K
value80Bit = value80Bit * quantity;
#endif
#if TARGET_CPU_PPC
    valueDouble = x80tod(&value80Bit);
    valueDouble = valueDouble * quantity;
    dtox80(&valueDouble,&value80Bit);
#endif

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

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

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doAcceptDateField

void  doAcceptDateField(TEHandle dateEditHdl)
{
    DateCacheRecord dateCacheRec;
    Ptr                   textPtr;
    SInt32              textLength, lengthUsed;
    LongDateRec           longDateTimeRec;
    LongDateTime        longDateTimeValue;
    Str255              dateString;
```

```
    InitDateCache(&dateCacheRec);
    textPtr = (*(*dateEditHdl)->hText);
    textLength = (*dateEditHdl)->teLength;

    HLock((*dateEditHdl)->hText);
    StringToDate(textPtr,textLength,&dateCacheRec,&lengthUsed,&longDateTimeRec);
    HUnlock((*dateEditHdl)->hText);

    LongDateToSeconds(&longDateTimeRec,&longDateTimeValue);
    longDateTimeValue += 15724800;
    LongDateString(&longDateTimeValue,longDate,dateString,NULL);
    doCopyPString(dateString,gLastRecordReviewDate);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doGetEditTextHandle

TEHandle  doGetEditTextHandle(ControlHandle controlHdl)
{
    TEHandle  textEditHdl;

    GetControlData(controlHdl,kControlNoPart,kControlEditTextTEHandleTag,
                   sizeof(textEditHdl),(Ptr) &textEditHdl,NULL);

    return textEditHdl;
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doDrawWindow

void  doDrawWindow(void)
{
    Rect theRect;

    SetRect(&theRect,144,198,310,300);
    EraseRect(&theRect);

    if(!gInBackground)
        SetThemeTextColor(kThemeTextColorModelessDialogActive,gPixelDepth,gIsColourDevice);
    else
        SetThemeTextColor(kThemeTextColorModelessDialogInactive,gPixelDepth,gIsColourDevice);

    doTodaysDate();

    MoveTo(32,39);
    DrawString("\pDate:");
    MoveTo(239,39);
    DrawString("\pTime:");

    MoveTo(34,99);
    DrawString("\pItem Title:");
    MoveTo(41,122);
    DrawString("\pQuantity:");
    MoveTo(32,145);
    DrawString("\pUnit Value:");
    MoveTo(61,168);
    DrawString("\pDate:");

    MoveTo(240,99);
    DrawString("\pEg: Barometers");
    MoveTo(240,122);
    DrawString("\pEg: 34");
    MoveTo(240,145);
    DrawString("\pEg: 135.58");
    MoveTo(240,168);
    DrawString("\pEg: 24 Jul 95");

    MoveTo(92,214);
    DrawString("\pItem Title:");
    MoveTo(99,234);
    DrawString("\pQuantity:");
    MoveTo(90,254);
    DrawString("\pUnit Value:");
    MoveTo(87,274);
    DrawString("\pTotal Value:");
    MoveTo(81,294);
    DrawString("\pReview Date:");

    MoveTo(112,349);
    DrawString("\pBalloon help is available");
```

```
  MoveTo(147,214);
  DrawString(gLastRecordItem);
  MoveTo(147,234);
  DrawString(gLastRecordQuantity);
  MoveTo(147,254);
  DrawString(gLastRecordUnitValue);
  MoveTo(147,274);
  DrawString(gLastRecordTotalValue);
  MoveTo(147,294);
  DrawString(gLastRecordReviewDate);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doAdjustCursor

void  doAdjustCursor(WindowPtr windowPtr)
{
  GrafPtr          oldPort;
  docStrucHandle  docStrucHdl;
  RgnHandle        arrowRegion,iBeamRegion;
  Rect             iBeamRect;
  Point            mouseXY;

  GetPort(&oldPort);
  SetPort(windowPtr);

  docStrucHdl = (docStrucHandle) GetWRefCon(windowPtr);

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

  SetRectRgn(arrowRegion,-32768,-32768,32766,32766);

  if(gCurrentEditStrucHdl == doGetEditTextHandle((*docStrucHdl)->itemControlHdl))
    iBeamRect = gItemRect;
  else if(gCurrentEditStrucHdl == doGetEditTextHandle((*docStrucHdl)->quantControlHdl))
    iBeamRect = gQuantRect;
  else if(gCurrentEditStrucHdl == doGetEditTextHandle((*docStrucHdl)->valueControlHdl))
    iBeamRect = gValueRect;
  else if(gCurrentEditStrucHdl == doGetEditTextHandle((*docStrucHdl)->dateControlHdl))
    iBeamRect = gDateRect;

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

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

  GetMouse(&mouseXY);
  LocalToGlobal(&mouseXY);

  if(PtInRgn(mouseXY,iBeamRegion))
  {
    SetCursor(*(GetCursor(iBeamCursor)));
    CopyRgn(iBeamRegion,gCursorRegion);
  }
  else
  {
    SetCursor(&qd.arrow);
    CopyRgn(arrowRegion,gCursorRegion);
  }

  DisposeRgn(arrowRegion);
  DisposeRgn(iBeamRegion);

  SetPort(oldPort);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doCopyPString

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

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

```
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doGetDepthAndDevice

void doGetDepthAndDevice(void)
{
  GDHandle deviceHdl;

  deviceHdl = LMGetMainDevice();
  gPixelDepth = (*(*deviceHdl)->gdPMap)->pixelSize;
  if(BitTst(&(*deviceHdl)->gdFlags,gdDevType))
    gIsColourDevice = true;
}

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

Demonstration Program Text2 Comments

When this program is run, the user should enter data in the four edit text fields, using the tab key or mouse clicks to select the required field and pressing the Return key when data has been entered in all fields.

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.

#define

The constants beginning with c represent resource IDs for group box and edit text field controls. kTab, kDel, and kReturn represent the character codes generated by the tab, forward-delete, and return keys. The following four constants are used to limit the number of characters that can be entered in a particular edit text field.

#typedef

The DocStruc data type will be used for a document structure, which will be attached to the single window opened by the program. The four fields of this structure will be assigned handles to separate edit structures.

Global Variables

gIsColourDevice and gPixelDepth will be used by the Appearance Manager function SetThemeTextColor. gCursorRegion relates to the cursorRgn parameter of the WaitNextEvent function.

gCurrentEditStrucHdl will be assigned the handle to the edit structure for the currently active edit text field. gMaxCharasThisField will be assigned a value representing the maximum number of characters allowed to be entered in the currently active edit text field. The four Rect variables will be used by the cursor shape changing function to establish the I-Beam cursor region for the currently active edit text field. The four Str255 variables will be assigned the strings to be drawn in the "last Record Entered" group box when the Return key is pressed.

main

SetThemeWindowBackground sets an Appearance-compliant colour/pattern for the window's content area. This means that the content area will be automatically repainted with that colour/pattern when required with no further assistance from the application.

doGetControls creates the controls for the window, following which the window is displayed and the background colour and pattern are saved.

doGetControls

doGetControls creates the window's controls.

Firstly, a root control is created for the window. The edit text field and group box controls are then created, their handles being assigned to the relevant field of the window's document structure.

At the next block, SetControlFontStyle is used to set the font for the edit text field controls to the small system font and the font for the "Last Record Entered" group box to the small emphasized system font.

At the last block, SetKeyboardFocus is called to set the keyboard focus to the first edit text field control, the associated edit structure is made the current edit structure, and the global variable which will limit the number of characters that may be entered in this edit text field is assigned the appropriate value.

eventLoop

When a NULL event is received, the application-defined function doIdle is called. Because IdleControls will be called in the idle function to blink the insertion point caret, WaitNextEvent's sleep parameter is assigned the value returned by a call to LMGetCaretTime

doIdle

doIdle is called when WaitNextEvent returns a null event. doIdle blinks the insertion point caret and draws the current time in the top right of the window.

IdleControls is called to ensure that the caret blinks regularly in the edit text field control with keyboard focus.

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 drawn in the window and the retrieved raw seconds value is assigned to the static variable oldRawSeconds for use next time doIdle is called.

doEvents

doEvents handles initial event processing.

In the case of a mouse-down event in the content region of the window (when it is the front window), the application-defined function doInContent is called.

In the case of a non-Command key equivalent key-down event, the application-defined function doKeyEvent is called.

doUpdate and doActivateWindow

doUpdate handles update events. Between the usual calls to BeginUpdate and EndUpdate, the contents of the window are re-drawn.

doActivateWindow activates or deactivates the window's contents. GetRootControl gets a handle to the window's root control. This is used in the following lines to either activate or deactivate all of the controls in the window. In addition, the application-defined function doDrawWindow is called to draw all the non-control text in the window in either the normal or dimmed state, as appropriate

doMenuChoice

doMenuChoice handles Apple, File, and Edit menu choices. In the case of choices from the Edit menu, the appropriate TextEdit function is called.

doAdjustMenus

doAdjustMenus adjusts the menus.

If there is a selection range in the current edit structure, the Cut, Copy, and Clear items are enabled, otherwise they are disabled. If there is any text in the TextEdit private scrap (the call to TEGetScrapLength), the Paste item is enabled, otherwise it is disabled. If there is any text in the currently activate edit text field, Select All is enabled, otherwise it is disabled.

doKeyEvent

doKeyEvent handles key-down and auto-key events which are not Command key equivalents.

If the character code is that generated by the tab key or the del key, handling is passed to other application-defined functions. If the Return key was pressed, control is passed to an application-defined function which accepts, formats, and displays the entered data and deletes all text from the edit text fields.

The following occurs if the character code makes it to the else block. Firstly, if the active edit text field is the Quantity or Unit Value field, and if the character code is not a numeric character, the minus character, or the period character, the system alert sound is played and the function returns. TEKey is called provided that the maximum number of characters allowable in the currently active edit text field will not be exceeded *or* the character is one of the non-printing characters. If the number of characters entered is at the limit and a printing character key was pressed, the character is not accepted and the system alert sound is played. This arrangement ensures that, if the number of characters in the edit structure is at the maximum allowed, the arrow and delete keys, unlike the printing character keys, will not be ignored and will have their usual effect.

doHandleTabKey

doHandleTabKey handles the tab key. Its purpose is to cycle around the edit text fields in response to tab key presses, defeating keyboard focus on the currently active edit text field and setting keyboard focus on the next edit text field in the sequence.

The first two lines retrieve a handle to the document structure for the window. The call to GetKeyboardFocus gets a handle to the control with keyboard focus. This will be used in the call to Draw1Control at the bottom of the function.

The central block determines which is the current edit structure (and thus the current edit text field) and calls the application-defined function that changes the active edit text field. This call passes the handle to the next edit text field control in the sequence, together with the maximum number of characters allowed in that field.

The call to Draw1Control on the "old" edit text field control ensures that the insertion point caret will be erased in that field. The call to doAdjustCursor is required in order to change the rectangle within which the cursor changes to the I-beam shape.

doHandleDelKey

doHandleDelKey handles the forward-delete key, which is not supported by TextEdit. The first line gets the current selection length. If the selection length is zero (that is, an insertion point caret is being displayed), the selection is set to include the character to the right of the insertion point. TEDelete deletes the current selection range from the text.

doInContent

doInContent handles mouse-down events in the content region of the window.

The second and third lines get a handle to the window's document structure. . The call to GetKeyboardFocus gets a handle to the control with keyboard focus. (This will be used in the call to Draw1Control at the bottom of the function.) The next line gets the position of shift key at the time of the mouse-down. (TEClick's behaviour depends on the position of the shift key.)

GlobalToLocal converts the global coordinates of the mouse-down to the local coordinates required by FindControl and TEClick.

If FindControl reports an enabled item under the mouse, and if that item is the Item Title edit text field control, and if that control has the keyboard focus, TEClick is called to take control until the mouse button is released. (Note that the shift key position is passed in the second parameter.) If the Item Title edit text field control does not have the keyboard focus, the application-defined function doChangeCurrentEditRec is called to change the keyboard focus to that control. The same basic procedure is followed in the case of mouse-downs within the other three edit text field controls.

The call to Draw1Control on the "old" edit text field control ensures that the insertion point caret will be erased in that field. The call to doAdjustCursor is required in order to change the rectangle within which the cursor changes to the I-beam shape.

doChangeCurrentEditRec

doChangeCurrentEditRec is called by both doHandleTabKey and doInContent to change the edit text field control with keyboard focus (and thus the currently active edit structure).

SetKeyboardFocus sets the keyboard focus to the specified edit text field control. The next line makes the edit structure associated with that control the current edit structure.

If this function was called by doInContent (teClick = true), TEClick is called to tell TextEdit that the mouse-down has occurred in the newly-activated edit structure. Note that the extend parameter is set to false, telling TextEdit, regardless of the position of the shift key, that the user is not extending a selection.

TESetSelect ensures that, if there are any characters in the edit structure, they will be initially highlighted. The last line assigns the appropriate value to the global variable which controls the maximum allowable number of characters in the active edit text field.

doTodaysDate

doTodaysDate draws the date at the top of the window.

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 drawn in the window.

doAcceptNewRecord

doAcceptNewRecord is called when the Return key is pressed. Assuming each edit text field contains at least one character of text, it calls other application-defined functions to format (where necessary) and display strings in the "Last Record Entered" group box area, and prepares the edit text fields to accept new data.

The first two lines get a handle to the window's document structure. The next block copies the handles to the four edit structures to four local variables.

At the next block, if any one of the edit text fields does not contain any text, the system alert sound is played and the function returns.

The two calls to GetDialogItemText get the text in the "Item Title" and "Quantity" edit text fields into two global variables. (GetDialogItemText is usually used to get the text from an editable edit text item in a dialog box. It works here because the first parameter required by GetDialogItemText is a handle to the hText field of an edit structure.) No special formatting of these two strings is required.

The call to InvalRect causes an update event to be generated. As will be seen, this causes the strings created by the preceding block to be drwan in the "Last Record Entered" group box area.

The next two lines call two application defined functions to format and display strings derived from the information in the "Unit Value" and "Date" edit text fields.

The next 10 lines delete the text from all edit text fields. The last three lines change the keyboard focus back to the "Item Title" edit text field.

doAcceptValueField

doAcceptValueField is called by doAcceptNewRecord to get the string from the "Value" edit structure, convert it to floating point number, convert that number to a formatted number string, draw that string, get the string in the "Quantity" edit structure, convert that string 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 draw that string .

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.

With that preparation attended to, GetDialogItemText gets the "Value" edit text field text into a Pascal string. StringToExtended converts that 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. doCopyPString copies that string to a global variable for subsequent drawing in the window.

The intention now is to multiply the quantity by the unit value to arrive at a total value. However, conditional compilation is required here to account for the fact that the PowerPC Floating Point Unit does not support Motorola's 80-bit extended type.

If the program is being compiled as 680x0 code, the value in the extended80 variable (unit value) is multiplied by the quantity and the result (total value) is stored in the extended80 variable. If the program is being compiled as PowerPC code, 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 copied to a global variable for subsequent drawing in the window.

doAcceptDateField

doAcceptDateField is called by doAcceptNewRecord to create a long date-time structure from the string in the "Date" edit structure, add six months to the date, format the date as a string (long date format), and draw that string in the structure panel.

The function which creates the long date-time structure takes an initialised date cache structure as a parameter. Accordingly, the first step is to initialise the specified date cache structure.

The next two lines get a pointer to the text in the "Date" edit structure and the length of that text. These are passed as parameters in the StringToDate call, which parses the input string and fills in the relevant fields of the long date-time structure.

LongDateToSeconds converts the long date-time structure to a long date-time value. The next line adds six months worth of seconds to the long date-time value. The modified long date-time value is then passed as a parameter to LongDateString. LongDateString 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 copied to a global variable for subsequent drawing in the window.

doGetEditTextHandle

doGetEditTextHandle returns the handle to the edit structure associated with the specified edit text control.

doDrawWindow

doDrawWindow draws the non-control text content of the window. The text will be drawn normal or dimmed depending on whether the application is in the foreground or background.

doAdjustCursor

doAdjustCursor is the cursor adjustment function. It is similar to the cursor adjustment functions in previous demonstration programs except that the rectangle in which the cursor is required to change to the I-beam cursor shape is changed to accord with that of the edit text field with keyboard focus.

Prior to the version issued with Mac OS 8.5 and later, the text backgound colour in the edit text control would often corrupt after cut and paste operations using the Edit menu (though not when using the Command-key equivalents). If you experience this problem when running the demonstration under a version of the Mac OS earlier than mac OS 8.5, proceed as follows:

Add this data type:

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

Add this global variable:

```
backColourPattern gBackColourPattern;
```

Add these function prototypes:

```
void doSaveBackground          (backColourPattern *);
void doRestoreBackground       (backColourPattern *);
void doSetBackgroundWhite      (void);
```

Add these functions:

```
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doSaveBackground

void  doSaveBackground(backColourPattern *gBackColourPattern)
{
   GrafPtr currentPort;

   GetPort(&currentPort);

   GetBackColor(&gBackColourPattern->backColour);
   gBackColourPattern->backPixelPattern    = NULL;

   if((**((CGrafPtr) currentPort)->bkPixPat).patType != 0)
      gBackColourPattern->backPixelPattern = ((CGrafPtr) currentPort)->bkPixPat;
   else
      gBackColourPattern->backBitPattern =
                                   *(PatPtr) (*(**((CGrafPtr) currentPort)->bkPixPat).patData);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doRestoreBackground

void  doRestoreBackground(backColourPattern *gBackColourPattern)
{
   RGBBackColor(&gBackColourPattern->backColour);

   if(gBackColourPattern->backPixelPattern)
      BackPixPat(gBackColourPattern->backPixelPattern);
   else
      BackPat(&gBackColourPattern->backBitPattern);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doSetBackgroundWhite

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

   RGBBackColor(&whiteColour);
   BackPat(&qd.white);
}
```

Add this call:

```
   doSaveBackground(&gBackColourPattern);
```

in the main function, after the call to ShowWindow.

Add this call:

```
   doRestoreBackground(&gBackColourPattern);
```

in doIdle, after call to IdleControls,
in doMenuChoice, just before the last break statement,
in doKeyEvent, just before the last closing brace,
in doIncontent, just before the last closing brace, and
in doChangeCurrentEditRec, as the first line.

Add this line:

```
   doSetBackgroundWhite();
```

in doMenuChoice, immediately before switch(menuItem),
in doKeyEvent, as the first line, and
in doInContent, as the first line.