

## ***MORE ON TEXT — MULTILINGUAL TEXT ENGINE***

***Demonstration Program: MLTETextEditor***

### ***Introduction***

---

The Multilingual Text Engine (MLTE), which was introduced with Mac OS 9.0 as an alternative for TextEdit, profoundly simplified the task of handling multistyled text, at the same time, provided many new features not provided by TextEdit. The main additional features are as follows:

- Documents can be larger than 32KB (the TextEdit limit).
- Tabs.
- Text justification.
- Ability to embed movie, sound, and graphics objects in documents.
- Built-in support for:
  - Scroll bar creation and handling, including live scrolling and proportional scroll boxes.
  - Undo/redo (32 levels).
  - Printing.
  - Drag and drop.
  - The public clipboard, specifically, copying and pasting:
    - Plain text.
    - Plain text with style resources.

- Unicode text.
- Flattened Unicode style information.
- Movies, graphics, and sound.
- In-line input.

MLTE uses Apple Type Services for Unicode Imaging (**ATSUI**) to measure and draw text. ATSUI, which was introduced with Mac OS 8.6, replaced QuickDraw and the Script Manager as the low-level means of imaging and measuring text.

MLTE renders text into a rectangular **frame**. Applications can specify that lines be arbitrarily wide or auto-wrapped.

---

## ***Global Layout Settings***

The following are the main settings applying to the whole document:

- Justification, which can be default, left, right, centre, full, or forced full.
- Tab values.
- Margins.
- Auto-indent on or off.
- Text auto-wrap on or off.
- Read-only status on or off.
- Line direction.

---

## ***Undoable and Re-doable Actions***

The following actions are undoable and re-doable:

- Typing.
- Cut, paste, and clear.
- Change font, and font size, style and colour.
- Justification.
- Drag and drop move and copy.

---

## ***Selection Behaviour***

Within an MLTE document, a single-click defines an insertion point. (Recall from Chapter 21 that an insertion point is, in effect, a selection containing zero characters.) A double-click selects a word, and a triple-click selects a line.

---

## ***File Types***

MLTE supports saving and opening files of the following types:

- A new type introduced with MLTE called **Textension** ('txtn'). This should be the preferred type for media-rich documents that can contain movies, graphics, and sound.
- Text ('TEXT'), with or without style information. Style information may be saved as either 'styl' resources or 'MPSR' resources. If 'styl' resources are used, documents can have text in an unlimited number of styles; however, tabs will not be saved. If 'MPSR' resources are used, only the first style in the document will be saved.<sup>1</sup>

---

<sup>1</sup> For example, SimpleText saves style information in 'styl' resources whereas Macintosh Programmer's Workshop (MPW), CodeWarrior and BBEdit save style information in MPW 'MPSR' resources.

- Plain Unicode text ('utxt').
- Movie ('MOOV'), sound ('Sfil' and 'AIFf'), and picture ('PICT').

## Working With MLTE

---

### Initialising and Terminating MLTE

---

TXNInitTextension should be called at program start to initialise the Textension library:

```
OSStatus TXNInitTextension(const TXNMacOSPreferredFontDescription iDefaultFonts[],
                          itemCount iDefaultFonts, TXNInitOptions iUsageFlags);
```

iDefaultFonts      NULL, or a table of entries for any encoding for which it is desired to designate a default font:

```
struct TXNMacOSPreferredFontDescription
{
    UInt32    fontID;    // Can assign kTXNDefaultFontName
    Fixed     pointSize; // Can assign kTXNDefaultFontSize
    TextEncoding encoding;
    Style     fontStyle; // Can assign kTXNDefaultFontStyle
};
```

For the encoding field, relevant constants are:

```
kTXNSystemDefaultEncoding
kTXNMacOSEncoding
kTXNUnicodeEncoding
```

kTXNSystemDefaultEncoding is the encoding preferred by MLTE and the system. This is Unicode if ATSUI is present, as it will be on Mac OS 8.6 and later.)

iDefaultFonts      Number of default fonts designated.

iUsageFlags        Specifies whether movies, sound, and/or graphics should be supported. Relevant constants are:

```
kTXNWantMoviesMask
kTXNWantSoundMask
kTXNWantGraphicsMask
```

At program termination, you should call TXNTerminateTextension to close the Textension library and perform other clean-up actions.

### Allocating and Deleting a TXNObject

---

Ordinarily, to create a new document, you create a new window and then pass a reference to that window in a call to TXNNewObject. TXNNewObject creates a new TXNObject, an object which contains private variables and functions required to handle text formatting:

```
OSStatus TXNNewObject(const FSSpec *            iFileSpec,    // Can be NULL
                     WindowRef            iWindow,
                     Rect *                iFrame,        // Can be NULL
                     TXNFrameOptions        iFrameOptions,
                     TXNFrameType          iFrameType,
                     TXNFileType          iFileType,
                     TXNPermanentTextEncodingType iPermanentEncoding,
                     TXNObject *          oTXNObject,
                     TXNFrameID *         oTXNFrameID,
                     TXNObjectRefcon        iRefCon);
```

iFileSpec            Pointer to file system specification structure. The file is read in, and its contents displayed, after the TXNObject is allocated. If NULL is passed in this parameter, the document will be empty at start.

iWindow             Reference to the window to be attached to the TXNObject, and in which the document will be displayed.

iFrame              The area of the window in which the document's contents are to be displayed. Passing NULL in this parameter sets the frame to equate to the window's port rectangle.

iFrameOptions        The options supported by this frame. The principal relevant constants are:

	<p>kTXNShowWindowMask Show window before TXNNewObject returns.</p> <p>kTXNWantHScrollBarMask Include horizontal scroll bar.</p> <p>kTXNWantVScrollBarMask Include vertical scroll bar.</p> <p>kTXNNoSelectionMask Do not display insertion point.</p>
iFrameType	<p>The frame type. Relevant constants are:</p> <p>kTXNTextEditStyleFrameType</p> <p>kTXNPageFrameType</p> <p>kTXNMultipleFrameType</p>
iFileType	<p>The primary file type. The principal relevant constants are:</p> <p>kTXNExtensionFile</p> <p>kTXNTextFile</p> <p>kTXNUnicodeTextFile</p> <p>If you specify kTXNTextFile, and want style information to be saved, you can specify whether that information should be saved in a 'styl' resource or an 'MPSR' resource when calling the function TXNSave (see below).</p>
iPermanentEncoding	<p>The encoding the application considers text to be in. Relevant constants are:</p> <p>kTXNSystemDefaultEncoding Encoding preferred by MLTE and the system. (This is Unicode if ATSUI is present.)</p> <p>kTXNMacOSEncoding Incoming and outgoing text to be in traditional MacOS encoding.</p> <p>kTXNUnicodeEncoding Incoming and outgoing text to be in Unicode, even on systems that do not have ATSUI.</p>
oTXNObject	On output, a pointer to a TXNObject.
oTXNFrameID	On output, a unique ID for the frame.
iRefCon	Reference constant for use by the application.

If TXNNewObject is called with NULL passed in the iWindow parameter, a window can later be attached to the TXNObject by a call to the function TXNAttachObjectToWindow.

A previously allocated TXNObject and its associated data structures may be deleted by a call to the function TXNDeleteObject.

## Setting and Getting Global Layout Settings

---

As previously stated, certain layout settings (for example, justification, tabs, and margins) apply to the whole TXNObject, that is, the whole document. These layout settings are referred to as **control information**. You can set control information by calling TXNSetTXNObjectControls:

```
OSStatus TXNSetTXNObjectControls(TXNObject iTXNObject,
                                Boolean iClearAll,
                                ItemCount iControlCount,
                                TXNControlTag iControlTags[],
                                TXNControlData iControlData[]);
```

iClearAll	Pass true to reset all controls to the default.
iControlCount	The number of elements in the iControlTags and iControlData arrays, that is, the number of settings being changed.
iControlTags	<p>An array of type TXNControlTag containing control tags. The principal relevant tags are:</p> <p>kTXNJustificationTag</p> <p>kTXNTabSettingsTag</p> <p>kTXNMarginsTag</p> <p>kTXNWordWrapStateTag</p>
iControlData	<p>An array of TXNControlData structures containing the control information being set. The TXNControlData structure, and its associated structures, are as follows:</p> <pre>union TXNControlData {</pre>

```

    UInt32    uValue;
    SInt32    sValue;
    TXNTab    tabValue;
    TXNMargins * marginsPtr;
};
typedef union TXNControlData;

struct TXNTab            struct TXNMargins
{                        {
    SInt16    value;        SInt16 topMargin;
    TXNTabType tabType;    SInt16 leftMargin;
    UInt8    filler;       SInt16 bottomMargin;
};                       SInt16 rightMargin;
typedef struct TXNTab TXNTab; };
                           typedef struct TXNMargins TXNMargins;

```

Constants relevant to the `uValue` field when setting justification are:

```

kTXNFlushDefault
kTXNFlushLeft
kTXNFlushRight
kTXNCenter
kTXNFullJust
kTXNForceFullJust

```

Constants relevant to the `tabType` field when setting tabs are:

```

kTXNRightTab
kTXNLeftTab
kTXNCenterTab

```

Constants relevant to the `uValue` field when setting word wrapping are:

```

kTXNAutoWrap
kTXNNoAutoWrap

```

You can get control information by calling `TXNGetTXNObjectControls`:

```

OSStatus TXNGetTXNObjectControls(TXNObject iTXNObject,
    NSInteger iControlCount,
    TXNControlTag iControlTags[],
    TXNControlData oControlData[]);

```

`iControlCount`     The number of elements in the `iControlTags` and `iControlData` arrays.

`iControlTags`     An array of type `TXNControlTag` containing control tags.

`oControlData`     An array of `TXNControlData` structures containing, on output, the control information requested by the tags in the `iControlTags` array.

## Setting the Background

---

You can set the background by calling `TXNSetBackground`:

```

OSStatus TXNSetBackground(TXNObject iTXNObject, TXNBackground * iBackgroundInfo);

```

`iBackgroundInfo`     A pointer to a `TXNBackground` structure which describes the background. The `TXNBackground` structure and its associated union are as follows:

```

struct TXNBackground
{
    TXNBackgroundType bgType; // Assign kTXNBackgroundTypeRGB
    TXNBackgroundData bg;
};
typedef struct TXNBackground TXNBackground;

union TXNBackgroundData
{
    RGBColor color;
};
typedef union TXNBackgroundData TXNBackgroundData;

```

The only background type available with Version 1.1 of MLTE is a colour. `TXNBackgroundData` is a union so that it can be expanded in the future to support other background types, such as pictures.

## Setting and Getting Type Attributes

---

You can set type attributes such as text size, style and colour by calling `TXNSetTypeAttributes`:

```
OSStatus TXNSetTypeAttributes(TXNObject      iTXNObject,
                             ItemCount      iAttrCount,
                             TXNTypeAttributes iAttributes[],
                             TXNOffset      iStartOffset,
                             TXNOffset      iEndOffset);
```

`iAttrCount` The number of elements in the `iAttributes` array, that is, the number of attributes being set.

`iAttributes` An array of `TXNTypeAttributes` structures specifying the attribute being set and the data, or pointer to the data, that will set the attribute. Values less than or equal to `sizeof(UInt32)` are passed by value. Values greater than `sizeof(UInt32)` are passed by pointer. The `TXNTypeAttributes` structure, and its main associated structure, are as follows:

```
struct TXNTypeAttributes
{
    TXNTag      tag;
    ByteCount   size;
    TXNAttributeData data;
};
typedef struct TXNTypeAttributes TXNTypeAttributes;

union TXNAttributeData
{
    void *      dataPtr;
    UInt32      dataValue;
    TXNATSUIFeatures * atsuFeatures;
    TXNATSUIVariations * atsuVariations;
};
typedef union TXNAttributeData TXNAttributeData;
```

The principal constants relevant to the `tag` field are:

```
kTXNQDFontSizeAttribute
kTXNQDFontStyleAttribute
kTXNQDFontColorAttribute
```

The associated constants relevant to the `size` field are:

```
kTXNFontSizeAttributeSize
kTXNQDFontStyleAttributeSize
kTXNQDFontColorAttributeSize
```

`iStartOffset` The offset at which to begin setting the attributes. If the requirement is to apply the attributes to the current selection, pass `kTXNUseCurrentSelection` in this parameter.

`iEndOffset` The offset at which to end setting the attributes. This parameter is ignored if `kTXNUseCurrentSelection` is passed in the `iStartOffset` parameter.

When your application detects a mouse-down in the menu bar or a Command-key combination, it typically adjusts its menus, enabling and disabling menu items as appropriate. If your application contains menus which allow the user to set type attributes, it must also prepare those menus for display by checkmarking and un-checkmarking items as appropriate.

Using a Size menu as an example, if the current selection (whether it be an empty or non-empty selection) contains text which is all of a single size, the menu item corresponding to that size, and only that menu item, should be checkmarked before the menu is displayed. However, if the selection contains text in two or more sizes, all menu items should be un-checkmarked. It is thus necessary to examine the selection to determine whether it contains a continuous size run or multiple sizes.

You can examine the current selection to determine whether font size, style, and colour are continuous by calling `TXNGetContinuousTypeAttributes`:

```
OSStatus TXNGetContinuousTypeAttributes(TXNObject      iTxnObject,
                                       TXNContinuousFlags * oContinuousFlags,
                                       ItemCount      iCount,
                                       TXNTypeAttributes ioTypeAttributes[]);
```

oContinuousFlags	On output, the relevant bit, or bits, of this parameter can be examined to determine whether the associated attribute, or attributes, is/are continuous. For example, if size is continuous, bit 1 will be set. The relevant constants are: kTXNSizeContinuousMask kTXNStyleContinuousMask kTXNColorContinuousMask
iCount	Number of elements in the ioTypeAttributes array, that is, the number of attributes being examined.
ioTypeAttributes	An array of TXNTypeAttributes structures (see above). On input, the tag field in each structure specifies the attribute to be examined. On output, if the attribute is continuous, the dataValue or dataPtr field of the data field will contain a value, or a pointer to data, which can be used to determine which menu item should be checkmarked.

## Functions Relevant to Events

The following functions are relevant to event handling:

<b>Function</b>	<b>Description</b>
TXNClick	Processes clicks in the content region, handling text selection, scrolling, drag and drop, and playing movies and sound.
TXNUpdate	Handles update events, redrawing the contents of the window. Calls BeginUpdate and EndUpdate.
TXNForceUpdate	Forces an update event to be generated, thus forcing the contents of the window to be redrawn.
TXNDraw	Similar to TXNUpdate, except that BeginUpdate and EndUpdate are not called. Should be used, in lieu of TXNUpdate, for a window that contains multiple TXNObjects or some graphic element.
TXNActivate	In MLTE, activation is a two-step process. TXNActivate performs the first step, which has to do with activating or deactivating the scroll bars. When true is passed in the TXNScrollBarState parameter, the scroll bars are activated, even when text input (selection and typing) has been defeated by TXNFocus (see below). When false is passed in the TXNScrollBarState parameter, the scroll bars are deactivated.
TXNFocus	The second step in the activation process has to do with activating text input (selection and typing). When true is passed in the iBecomingFocused parameter, text input is activated. When false is passed in the iBecomingFocused parameter text input is deactivated.
TXNAdjustCursor	Handles cursor shape-changing. If the cursor is over a text area, it is set to the I-beam shape. If it is over a scrollbar, a movie, graphic, or a sound, or outside the frame, it is set to the arrow shape. Not relevant in applications using the Carbon event model.
TXNZoomWindow	Handles mousedown in the zoom box.
TXNGrowWindow	Handles mouse-downs in the size box.
TXNGetSleepTicks	Gets the appropriate sleep time.

Note that, in Carbon applications, calling TXNKeyDown on receipt of key events is not necessary, since MLTE handles the event itself without any assistance on the part of your application. Note also that:

- When the Classic event model is used, it is not possible to filter key events because the event is sent to MLTE before WaitNextEvent delivers it to your application.
- When the Carbon event model is used, key events are sent through the Carbon event system before they are sent to MLTE. This means that you can filter key events by, for example, installing a handler for the kEventUnicodeForKeyEvent event kind (kEventClassTextInput event class).

## Functions Relevant to the File Menu

The following functions are relevant to your application's File menu:

<b>Function</b>	<b>Description</b>
TXNSave	Saves the contents of a document to a file of a specified file type (for example, Textension, text, or



	Unicode text). The data fork of the file must be open and its file reference number passed in the <code>iDataReference</code> parameter. For files of type text, if style information is also to be saved, the resource fork of the file must also be open and its file reference number passed in the <code>iResourceReference</code> parameter (The <code>iResourceReference</code> parameter is ignored when the Textension file type is specified.) <code>kTXNMultipleStylesPerTextDocumentResType</code> passed in the <code>iResType</code> parameter causes style information be saved to a 'style' resource. To specify that style information be saved to an 'MPSR' resource, pass <code>kTXNSingleStylePerTextDocumentResType</code> in the <code>iResType</code> parameter. <code>TXNSave</code> does not move the file mark before writing to a file. This allows you to write private data first (if required), followed by the data written by <code>TXNSave</code> .
<code>TXNRevert</code>	Reverts to the last saved version of the document or, if the document has never been saved, reverts to an empty document.
<code>TXNPageSetup</code>	Displays the Page Setup dialog and reformats the text if settings are changed by the user.
<code>TXNPrint</code>	Displays the Print dialog and prints the document.
<code>TXNGetChangeCount</code>	Returns the number of times the document has been changed since the last save or, for new documents which have not yet been saved, since the document was created. Useful for determining whether the Save and Revert items should be enabled or disabled.
<code>TXNDataSize</code>	Returns the size in bytes of the characters in the document. Useful for determining if the Save As, Page Setup and Print items should be enabled or disabled.

## Functions Relevant to the Edit Menu

The following functions are relevant to your application's Edit menu:

<b>Function</b>	<b>Description</b>
<code>TXNCanUndo</code>	Returns true if the last action is undoable, in which case the Undo item should be enabled.
<code>TXNCanRedo</code>	Returns true if the last action is re-doable, in which case the Redo item should be enabled.
<code>TXNUndo</code>	Undoes the last action.
<code>TXNRedo</code>	Re-does the last action.
<code>TXNCut</code>	Cuts the current selection to the MLTE private scrap.
<code>TXNCopy</code>	Copies the current selection to the MLTE private scrap.
<code>TXNPaste</code>	Pastes the MLTE private scrap to the document.
<code>TXNClear</code>	Clears the current selection.
<code>TXNSelectAll</code>	Selects everything in a frame.
<code>TXNIsSelectionEmpty</code>	Returns false if the current selection is not empty, in which case the Cut, Copy, and Clear items should be enabled.
<code>TXNIsScrapPastable</code>	Returns true if the current MLTE scrap is pastable, in which case the Paste item should be enabled.
<code>TXNConvertToPublicScrap</code>	Converts the MLTE private scrap to the public clipboard. Note that, in Carbon applications, this function should not be called on suspend events. Typically, it should be called immediately after <code>TXNCut</code> and <code>TXNCopy</code> .
<code>TXNDataSize</code>	Returns the size in bytes of the characters in the document. Useful for determining if the Select All item should be enabled or disabled.

Note that, in Carbon applications, there is no need to call `TXNConvertFromPublicScrap` to convert the public clipboard to the MLTE private scrap. In Carbon applications, MLTE automatically keeps the public scrap and private scrap synchronised.

### More on `TXNCanUndo` and `TXNCanRedo`

When `TXNCanUndo` and `TXNCanRedo` return true, they also return in their `oTXNActionKey` parameters an **action key** which can be used by your application to load an indexed string describing the undoable or re-doable action. The Undo and Redo items should be set to this string using `SetMenuItemText`. The following are the main action key constants:

<code>kTXNTypingAction</code>	<code>kTXNChangeFontColorAction</code>	<code>kTXNAlignLeftAction</code>
<code>kTXNCutAction</code>	<code>kTXNChangeFontSizeAction</code>	<code>kTXNAlignRightAction</code>
<code>kTXNPasteAction</code>	<code>kTXNChangeStyleAction</code>	<code>kTXNAlignCenterAction</code>

kTXNClearAction      kTXNMoveAction      kTXNUndoLastAction  
kTXNChangeFontAction      kTXNDropAction

## Creating, Preparing, and Handling the Font Menu

---

MLTE contains functions which greatly simplify the task of creating and managing the Font menu.

The TXNNewFontMenuObject function may be used to create a hierarchical Font menu. A reference to the (empty) Font menu, the menu ID, and the starting ID for the sub-menus are passed in the first three parameters, and a pointer to a TXNFontMenuObject is returned in the fourth parameter. The starting ID for the sub-menus must be 160 or higher.

To prepare the Font menu for display when the user clicks in the menu bar, you should simply call TXNPrepareFontMenu, which checkmarks and un-checkmarks items in the Font menu as appropriate.

A call to TXNDoFontMenuSelection, with the TXNFontMenuObject obtained by the call to TXNNewFontMenuObject passed in the second parameter and the menu ID and chosen menu item passed in the third and fourth parameters, sets the chosen font and changes the current selection to that font.

At program termination, you should call TXNDisposeFontMenuObject to dispose of the TXNFontMenuObject and its menu handle.

## Setting Data

---

You can replace a specified range with data by calling TXNSetData:

```
OSStatus TXNSetData(TXNObject iTXNObject, TXNDataType iDataType, void * iDataPtr,  
ByteCount iDataSize, TXNOffset iStartOffset, TXNOffset iEndOffset);
```

iDataType      The type of data. Relevant constants are:

kTXNTextData  
kTXNPictureData  
kTXNMovieData  
kTXNSoundData  
kTXNUnicodeTextData

iDataPtr      A pointer to the new data.

iDataSize      The size of new data.

iStartOffset      The offset to the beginning of the range to replace.

iEndOffset      The offset to the end of range to replace.

You can replace a specified range with the contents of a specified file by calling TXNSetDataFromFile:

```
OSStatus TXNSetDataFromFile(TXNObject iTXNObject, SInt16 iFileRefNum, OSType iFileType,  
ByteCount iFileLength, TXNOffset iStartOffset,  
TXNOffset iEndOffset);
```

iFileRefNum      The file reference number.

iFileType      The file type. For file types supported by MLTE, the relevant constants are:

kTXNExtensionFile  
kTXNTextFile  
kTXNPictureFile  
kTXNMovieFile  
kTXNSoundFile  
kTXNAIFFFile  
kTXNUnicodeTextFile

iFileLength      How much data should be read. Ignored if the file type is a file type that MLTE supports.

This parameter is useful when you want data that is embedded in the file. For the whole file, pass kTXNEndOffset in this parameter

iStartOffset      The offset to the beginning of the range to replace.

iEndOffset      The offset to the end of range to replace.

The data fork of the file must be opened, and the file mark set, by the application. MLTE does not move the file's marker before reading the data.

# Main Constants, Data Types, and Functions

---

## Constants

---

### Initializing

kTXNWantMoviesMask = 1L << kTXNWantMoviesBit  
kTXNWantSoundMask = 1L << kTXNWantSoundBit  
kTXNWantGraphicsMask = 1L << kTXNWantGraphicsBit

### Text Encoding

kTXNSystemDefaultEncoding = 0  
kTXNMacOSEncoding = 1  
kTXNUnicodeEncoding = 2

### Frame Options

kTXNShowWindowMask = 1L << kTXNShowWindowBit  
kTXNWantHScrollBarMask = 1L << kTXNWantHScrollBarBit  
kTXNWantVScrollBarMask = 1L << kTXNWantVScrollBarBit  
kTXNNoSelectionMask = 1L << kTXNNoSelectionBit

### Frame Types

kTXNTextEditStyleFrameType = 1  
kTXNPageFrameType = 2  
kTXNMultipleFrameType = 3

### Global Layout

kTXNJustificationTag = FOUR\_CHAR\_CODE('just')  
kTXNTabSettingsTag = FOUR\_CHAR\_CODE('tabs')  
kTXNMarginsTag = FOUR\_CHAR\_CODE('marg')  
kTXNWordWrapStateTag = FOUR\_CHAR\_CODE('wwrs')  
kTXNFlushDefault = 0  
kTXNFlushLeft = 1  
kTXNFlushRight = 2  
kTXNCenter = 4  
kTXNFullJust = 8  
kTXNForceFullJust = 16  
kTXNAutoWrap = false  
kTXNNoAutoWrap = true  
kTXNRightTab = -1  
kTXNLeftTab = 0  
kTXNCenterTab = 1

### Setting the Background

kTXNBackgroundTypeRGB = 1

### Type Attributes

kTXNSizeContinuousMask = 1L << kTXNSizeContinuousBit  
kTXNStyleContinuousMask = 1L << kTXNStyleContinuousBit  
kTXNColorContinuousMask = 1L << kTXNColorContinuousBit  
kTXNQDFontSizeAttribute = FOUR\_CHAR\_CODE('size')  
kTXNQDFontStyleAttribute = FOUR\_CHAR\_CODE('face')  
kTXNQDFontColorAttribute = FOUR\_CHAR\_CODE('klor')  
kTXNFontSizeAttributeSize = sizeof(Fixed)  
kTXNQDFontStyleAttributeSize = sizeof(Style)  
kTXNQDFontColorAttributeSize = sizeof(RGBColor)

### Data Types

kTXNTextData = FOUR\_CHAR\_CODE('TEXT')  
kTXNPictureData = FOUR\_CHAR\_CODE('PICT')  
kTXNMovieData = FOUR\_CHAR\_CODE('moov')  
kTXNSoundData = FOUR\_CHAR\_CODE('snd ')  
kTXNUnicodeTextData = FOUR\_CHAR\_CODE('utxt')

### File Types

kTXNTextensionFile = FOUR\_CHAR\_CODE('txtn')  
kTXNTextFile = FOUR\_CHAR\_CODE('TEXT')  
kTXNPictureFile = FOUR\_CHAR\_CODE('PICT')  
kTXNMovieFile = FOUR\_CHAR\_CODE('MooV')  
kTXNSoundFile = FOUR\_CHAR\_CODE('sfil')  
kTXNAIFFFile = FOUR\_CHAR\_CODE('AIFF')

```
kTXNUnicodeTextFile = FOUR_CHAR_CODE('utxt')
```

## **Undo/Redo Action Keys**

```
kTXNTypingAction = 0
kTXNCutAction = 1
kTXNPasteAction = 2
kTXNClearAction = 3
kTXNChangeFontAction = 4
kTXNChangeFontColorAction = 5
kTXNChangeFontSizeAction = 6
kTXNChangeStyleAction = 7
kTXNAlignLeftAction = 8
kTXNAlignCenterAction = 9
kTXNAlignRightAction = 10
kTXNDropAction = 11
kTXNMoveAction = 12
kTXNUndoLastAction = 1024
```

## **Saving Text Files — Style Information**

```
kTXNSingleStylePerTextDocumentResType = FOUR_CHAR_CODE('MPSR')
kTXNMultipleStylesPerTextDocumentResType = FOUR_CHAR_CODE('styl')
```

## **Data Types**

---

```
typedef struct OpaqueTXNObject* TXNObject;
typedef struct OpaqueTXNFontMenuObject* TXNFontMenuObject;
typedef UInt32 TXNFrameID;
typedef OptionBits TXNInitOptions;
typedef UInt32 TXNPermanentTextEncodingType;
typedef OptionBits TXNFrameOptions;
typedef UInt32 TXNFrameType;
typedef UInt32 TXNBackgroundType;
typedef FourCharCode TXNControlTag;
typedef SInt8 TXNTabType;
typedef OptionBits TXNContinuousFlags;
typedef ByteCount TXNTypeRunAttributeSizes;
typedef OSType TXNDataType;
typedef OSType TXNFileType;
typedef UInt32 TXNActionKey;
```

## **Global Layout**

```
union TXNControlData
{
    UInt32 uValue;
    SInt32 sValue;
    TXNTab tabValue;
    TXNMargins * marginsPtr;
};
typedef union TXNControlData TXNControlData;
```

```
struct TXNTab
{
    SInt16 value;
    TXNTabType tabType;
    UInt8 filler;
};
typedef struct TXNTab TXNTab;
```

```
struct TXNMargins
{
    SInt16 topMargin;
    SInt16 leftMargin;
    SInt16 bottomMargin;
    SInt16 rightMargin;
};
typedef struct TXNMargins TXNMargins;
```

## **Setting the Background**

```
struct TXNBackground
{
    TXNBackgroundType bgType;
    TXNBackgroundData bg;
};
typedef struct TXNBackground TXNBackground;
```

```
union TXNBackgroundData
{
```

```

    RGBColor color;
};
typedef union TXNBackgroundData TXNBackgroundData;

```

## **Type Attributes**

```

struct TXNTypeAttributes
{
    TXNTag tag;
    ByteCount size;
    TXNAttributeData data;
};
typedef struct TXNTypeAttributes TXNTypeAttributes;

```

```

union TXNAttributeData
{
    void * dataPtr;
    UInt32 dataValue;
    TXNATSUIFeatures * atsuFeatures;
    TXNATSUIVariations * atsuVariations;
};
typedef union TXNAttributeData TXNAttributeData;

```

## **Font Description**

```

struct TXNMacOSPreferredFontDescription
{
    UInt32 fontID;
    Fixed pointSize;
    TextEncoding encoding;
    Style fontStyle;
};
typedef struct TXNMacOSPreferredFontDescription TXNMacOSPreferredFontDescription;

```

## **Functions**

---

### **Initialising and Terminating**

```

void TXNTerminateTextension(void);
OSStatus TXNInitTextension(const TXNMacOSPreferredFontDescription iDefaultFonts[],
    itemCount iCountDefaultFonts, TXNInitOptions iUsageFlags);

```

### **Allocating and Deleting TXNObject**

```

OSStatus TXNNewObject(const FSSpec *iFileSpec, WindowRef iWindow, Rect *iFrame,
    TXNFrameOptions iFrameOptions, TXNFrameType iFrameType, TXNFileType iFileType,
    TXNPermanentTextEncodingType iPermanentEncoding, TXNObject *oTXNObject,
    TXNFrameID *oTXNFrameID, TXNObjectRefcon iRefCon);
void TXNDeleteObject(TXNObject iTXNObject);

```

### **Attaching an Object to a Window**

```

OSStatus TXNAttachObjectToWindow(TXNObject iTXNObject, GWorldPtr iWindow,
    Boolean iIsActualWindow);
Boolean TXNIsObjectAttachedToWindow(TXNObject iTXNObject);

```

### **Resizing the Frame**

```

void TXNResizeFrame(TXNObject iTXNObject, UInt32 iWidth, UInt32 iHeight,
    TXNFrameID iTXNFrameID);

```

### **Setting and Getting Global Layout Settings**

```

OSStatus TXNSetTXNObjectControls(TXNObject iTXNObject, Boolean iClearAll,
    itemCount iControlCount, TXNControlTag iControlTags[], TXNControlData iControlData[]);
OSStatus TXNGetTXNObjectControls(TXNObject iTXNObject, itemCount iControlCount,
    TXNControlTag iControlTags[], TXNControlData oControlData[]);

```

### **Setting the Background**

```

OSStatus TXNSetBackground(TXNObject iTXNObject, TXNBackground *iBackgroundInfo);

```

### **Setting and Getting Type Attributes**

```

OSStatus TXNSetTypeAttributes(TXNObject iTXNObject, itemCount iAttrCount,
    TXNTypeAttributes iAttributes[], TXNOffset iStartOffset, TXNOffset iEndOffset);
OSStatus TXNGetContinuousTypeAttributes(TXNObject iTxnObject,
    TXNContinuousFlags *oContinuousFlags, itemCount iCount,
    TXNTypeAttributes ioTypeAttributes[]);

```

### **Event Handling**

```

void TXNClick(TXNObject iTXNObject, const EventRecord *iEvent);

```

```

void TXNUpdate(TXNObject iTXNObject);
void TXNForceUpdate(TXNObject iTXNObject);
void TXNDraw(TXNObject iTXNObject,GWorldPtr iDrawPort);
OSStatus TXNActivate(TXNObject iTXNObject,TXNFrameID iTXNFrameID,
    TXNScrollBarState iActiveState);
void TXNFocus(TXNObject iTXNObject,Boolean iBecomingFocused);
void TXNAdjustCursor(TXNObject iTXNObject,RgnHandle ioCursorRgn);
void TXNZoomWindow(TXNObject iTXNObject,short iPart);
void TXNGrowWindow(TXNObject iTXNObject,const EventRecord *iEvent);
UInt32 TXNGetSleepTicks(TXNObject iTXNObject);

```

### **Functions Relevant to the File Menu**

```

OSStatus TXNSave(TXNObject iTXNObject,TXNFileType iType,OSType iResType,
    TXNPermanentTextEncodingType iPermanentEncoding,FSSpec *iFileSpecification,
    SInt16 iDataReference,SInt16 iResourceReference);
OSStatus TXNRevert(TXNObject iTXNObject);
OSStatus TXNPageSetup(TXNObject iTXNObject);
OSStatus TXNPrint(TXNObject iTXNObject);

```

### **Functions Relevant to the Edit Menu**

```

Boolean TXNCanUndo(TXNObject iTXNObject,TXNActionKey *oTXNActionKey);
Boolean TXNCanRedo(TXNObject iTXNObject,TXNActionKey *oTXNActionKey);
void TXNUndo(TXNObject iTXNObject);
void TXNRedo(TXNObject iTXNObject);
OSStatus TXNCut(TXNObject iTXNObject);
OSStatus TXNCopy(TXNObject iTXNObject);
OSStatus TXNPaste(TXNObject iTXNObject);
OSStatus TXNClear(TXNObject iTXNObject);
void TXNSelectAll(TXNObject iTXNObject);

```

### **Creating, Preparing, Handling, and Disposing of the Font Menu**

```

OSStatus TXNNewFontMenuObject(MenuRef iFontMenuHandle,SInt16 iMenuID,
    SInt16 iStartHierMenuID,TXNFontMenuObject *oTXNFontMenuObject);
OSStatus TXNPrepareFontMenu(TXNObject iTXNObject,TXNFontMenuObject iTXNFontMenuObject);
OSStatus TXNDoFontMenuSelection(TXNObject iTXNObject,TXNFontMenuObject iTXNFontMenuObject,
    SInt16 iMenuID,SInt16 iMenuItem);
OSStatus TXNGetFontMenuRef(TXNFontMenuObject iTXNFontMenuObject,MenuRef *oFontMenuHandle);
OSStatus TXNDisposeFontMenuObject(TXNFontMenuObject iTXNFontMenuObject);

```

### **Selections**

```

Boolean TXNIsSelectionEmpty(TXNObject iTXNObject);
void TXNGetSelection(TXNObject iTXNObject,TXNOffset *oStartOffset,TXNOffset *oEndOffset);
void TXNShowSelection(TXNObject iTXNObject,Boolean iShowEnd);
OSStatus TXNSetSelection(TXNObject iTXNObject,TXNOffset iStartOffset,TXNOffset iEndOffset);

```

### **Setting Data**

```

ByteCount TXNDataSize(TXNObject iTXNObject);
OSStatus TXNGetData(TXNObject iTXNObject,TXNOffset iStartOffset,TXNOffset iEndOffset,
    Handle *oDataHandle);
OSStatus TXNSetDataFromFile(TXNObject iTXNObject,SInt16 iFileRefNum,OSType iFileType,
    ByteCount iFileLength,TXNOffset iStartOffset,TXNOffset iEndOffset);

```

### **Getting the Change Count**

```

ItemCount TXNGetChangeCount(TXNObject iTXNObject);

```

### **Scrap**

```

Boolean TXNIsScrapPastable(void);
OSStatus TXNConvertToPublicScrap(void);

```

### **Font Defaults**

```

OSStatus TXNSetFontDefaults(TXNObject iTXNObject,ItemCount iCount,
    TXNMacOSPreferredFontDescription iFontDefaults[]);
OSStatus TXNGetFontDefaults(TXNObject iTXNObject,ItemCount *ioCount,
    TXNMacOSPreferredFontDescription iFontDefaults[]);

```

## Demonstration Program MLTETextEditor Listing

---

```
// *****
// MLETextEditor.h                                CLASSIC EVENT MODEL
// *****
//
// This program demonstrates the use of the Multilingual Text Engine API to create a basic
// multi-styled text editor. New documents created by the program are created and saved as
// Textension ('txtn') documents. Existing 'TEXT' documents and Unicode ('utxt') documents
// are saved in the original format. In the case of 'TEXT' documents, style information is
// saved in a 'styl' resource.
//
// The program utilises the following resources:
//
// • A 'plst' resource.
//
// • An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit, Size, Style, Colour,
//   and Justification (preload, non-purgeable).
//
// • A 'WIND' resource (purgeable) (initially not visible).
//
// • A 'STR ' resource (purgeable) containing the "missing application name" string, which is
//   copied to all document files created by the program.
//
// • 'STR#' resources (purgeable) containing error strings, the application's name (for
//   certain Navigation Services functions), and strings for the Edit menu Undo and Redo
//   items.
//
// • A 'kind' resource (purgeable) describing file types, which is used by Navigation
//   Services to build the native file types section of the Show pop-up menu in the Open
//   dialog box.
//
// • An 'open' resource (purgeable) containing the file type list for the Open dialog box.
//
// • The 'BNDL' resource (non-purgeable), 'FREF' resources (non-purgeable), signature
//   resource (non-purgeable), and icon family resources (purgeable), required to support the
//   built application.
//
// • A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//   doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// *****

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

#include <Carbon.h>

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

#define rMenuBar          128
#define mAppleApplication 128
#define iAbout           1
#define mFile            129
#define iNew             1
#define iOpen            2
#define iClose           4
#define iSave            5
#define iSaveAs          6
#define iRevert          7
#define iPageSetup       9
#define iPrint           10
#define iQuit            12
#define mEdit            130
#define iUndo            1
#define iRedo            2
#define iCut              4
#define iCopy            5
#define iPaste           6
#define iClear           7
#define iSelectAll       8
#define mFont            131
#define mSize            132
#define iTwelve          4
```



```

#define mStyle          133
#define iPlain          1
#define iBold           3
#define iUnderline     5
#define mColour        134
#define iBlack         4
#define iColourPicker  6
#define mJustification 135
#define iDefault       1
#define iLeft          2
#define iForceFull     6
#define mWindow        136
#define mFirstHierarchical 160

#define rNewWindow     128
#define rAboutDialog   128
#define rErrorStrings  128
#define eInstallHandler 1000
#define eMaxWindows    1001
#define eCantFindFinderProcess 1002
#define rMiscellaneousStrings 129
#define sApplicationName 1
#define rOpenResource  128

#define kMaxWindows    8
#define kOpen          0
#define kPrint         1
#define kFileCreator   'bbjk'
#define MAX_UINT32     0xFFFFFFFF
#define MIN(a,b)       ((a) < (b) ? (a) : (b))
#define topLeft(r)     (((Point *) &(r))[0])

#define kATSUCGContextTag 32767L

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

void    main                (void);
void    doPreliminaries     (void);
void    doInitialiseMTLE    (void);
void    doInstallAEHandlers (void);
void    eventLoop           (void);
UInt32  doGetSleepTime     (void);
void    doIdle              (void);
void    doEvents            (EventRecord *);
void    doMouseDown         (EventRecord *);
void    doBringFinderToFront (void);
OSStatus doFindProcess      (OSType,OSType,ProcessSerialNumber *);
void    doActivate          (EventRecord *);
void    doUpdate            (EventRecord *);
Boolean isApplicationWindow (WindowRef,TXNObject *);
void    doAboutDialog       (void);
void    doSynchroniseFiles  (void);
OSStatus openAppEventHandler (AppleEvent *,AppleEvent *,SInt32);
OSStatus reopenAppEventHandler (AppleEvent *,AppleEvent *,SInt32);
OSStatus openAndPrintDocsEventHandler (AppleEvent *,AppleEvent *,SInt32);
OSStatus quitAppEventHandler (AppleEvent *,AppleEvent *,SInt32);
OSStatus doHasGotRequiredParams (AppleEvent *);
void    doErrorAlert        (SInt16);
void    doCopyPString       (Str255,Str255);
void    doConcatPStrings    (Str255,Str255);

void    doEnableDisableMenus (Boolean);
void    doAdjustAndPrepareMenus (void);
void    doAdjustFileMenu     (MenuRef,WindowRef);
void    doAdjustEditMenu     (MenuRef,WindowRef);
void    doPrepareFontMenu    (WindowRef);
void    doPrepareSizeMenu    (MenuRef,WindowRef);
void    doPrepareStyleMenu   (MenuRef,WindowRef);
void    doPrepareColourMenu  (MenuRef,WindowRef);
Boolean isEqualRGB          (RGBColor *,RGBColor *);
void    doPrepareJustificationMenu (MenuRef,WindowRef);
void    doMenuChoice         (SInt32);
void    doFileMenuChoice     (MenuItemIndex,WindowRef);
void    doEditMenuChoice     (MenuItemIndex,WindowRef);
void    doFontMenuChoice     (MenuItemID,MenuItemIndex,WindowRef);
void    doSizeMenuChoice     (MenuItemIndex,WindowRef);
void    doStyleMenuChoice    (MenuItemIndex,WindowRef);

```

```

void doColourMenuChoice (MenuItemIndex,WindowRef);
void doJustificationMenuChoice (MenuItemIndex,WindowRef);

OSStatus doNewCommand (void);
OSStatus doOpenCommand (void);
OSStatus doCloseCommand (NavAskSaveChangesAction);
OSStatus doSaveCommand (void);
OSStatus doSaveAsCommand (void);
OSStatus doRevertCommand (void);
OSStatus doQuitCommand (NavAskSaveChangesAction);
OSStatus doNewDocWindow (WindowRef *,FSSpec *,TXNFileType);
OSStatus doOpenFile (FSSpec,OSType);
void doCloseWindow (WindowRef,TXNObject);
OSStatus doWriteFile (WindowRef,Boolean);
OSStatus doCopyResources (FSSpec,TXNFileType,Boolean);
OSStatus doCopyAResource (ResType,SInt16,SInt16,SInt16);
void navEventFunction (NavEventCallbackMessage,NavCBRecPtr,
NavCallBackUserData);

// *****
// MLTETextEditor.c
// *****

//
..... includes

#include "MLTETextEditor.h"

//
..... global variables

SInt16 gAppResFileRefNum;
Boolean gRunningOnX = false;
Boolean gDone;
TXNFontMenuObject gTXNFontMenuObject;
RgnHandle gCursorRgnHdl;
extern SInt16 gCurrentNumberOfWindows;

// ***** main

void main(void)
{
MenuBarHandle menubarHdl;
SInt32 response;
MenuRef menuRef;
OSStatus osStatus = noErr;

//
..... do preliminaries

doPreliminaries();

// ..... save application's resource file file reference number

gAppResFileRefNum = CurResFile();

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

menubarHdl = GetNewMBar(rMenubar);
if(menubarHdl == NULL)
doErrorAlert(MemError());
SetMenuBar(menubarHdl);

CreateStandardWindowMenu(0,&menuRef);
SetMenuID(menuRef,mWindow);
InsertMenu(menuRef,0);
DeleteMenuItem(menuRef,1);

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

```

```

    DeleteMenuItem(menuRef,iQuit);
    DeleteMenuItem(menuRef,iQuit - 1);
}

gRunningOnX = true;
}

// ..... build hierarchical font menu and draw
menu bar

menuRef = GetMenuRef(mFont);
osStatus = TXNNewFontMenuObject(menuRef,mFont,mFirstHierarchical,&gTXNFontMenuObject);
if(osStatus != noErr)
    doErrorAlert(osStatus);

DrawMenuBar();

// ..... install required
Apple event handlers

doInstallAEHandlers();

//
..... enter event loop

eventLoop();
}

// ***** doPreliminaries

void doPreliminaries(void)
{
    MoreMasterPointers(960);
    InitCursor();
    FlushEvents(everyEvent,0);

    doInitialiseMTLE();
}

// ***** doInitializeMTLE

void doInitialiseMTLE(void)
{
    TXNMacOSPreferredFontDescription defaultFont[1];
    OSStatus          osStatus = noErr;
    SInt16            fontID;

    GetFNum("\pNew York",&fontID);

    defaultFont[0].fontID = fontID;
    defaultFont[0].pointSize = 0x000C0000;
    defaultFont[0].fontStyle = kTXNDefaultFontStyle;
    defaultFont[0].encoding = kTXNSystemDefaultEncoding;

    osStatus = TXNInitTextension(&defaultFont[0],1,kTXNWantMoviesMask);
    if(osStatus != noErr)
        doErrorAlert(osStatus);
}

// ***** doInstallAEHandlers

void doInstallAEHandlers(void)
{
    OSStatus osStatus = noErr;

    osStatus = AEInstallEventHandler(kCoreEventClass,kAEOpenApplication,
        NewAEEEventHandlerUPP((AEEEventHandlerProcPtr) openAppEventHandler),
        0L,false);
    if(osStatus != noErr) doErrorAlert(eInstallHandler);

    osStatus = AEInstallEventHandler(kCoreEventClass,kAEReopenApplication,
        NewAEEEventHandlerUPP((AEEEventHandlerProcPtr) reopenAppEventHandler),
        0L,false);
    if(osStatus != noErr) doErrorAlert(eInstallHandler);

    osStatus = AEInstallEventHandler(kCoreEventClass,kAEOpenDocuments,
        NewAEEEventHandlerUPP((AEEEventHandlerProcPtr) openAndPrintDocsEventHandler),
        kOpen,false);
}

```

```

if(osStatus != noErr) doErrorAlert(eInstallHandler);

osStatus = AEInstallEventHandler(kCoreEventClass,kAEPrintDocuments,
    NewAEEEventHandlerUPP((AEEEventHandlerProcPtr) openAndPrintDocsEventHandler),
    kPrint,false);
if(osStatus != noErr) doErrorAlert(eInstallHandler);

osStatus = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
    NewAEEEventHandlerUPP((AEEEventHandlerProcPtr) quitAppEventHandler),
    0L,false);
if(osStatus != noErr) doErrorAlert(eInstallHandler);
}

// ***** eventLoop

void eventLoop(void)
{
    EventRecord eventStructure;

    gDone = false;
    gCursorRgnHdl = NewRgn();

    while(!gDone)
    {
        if(WaitNextEvent(everyEvent,&eventStructure,doGetSleepTime(),gCursorRgnHdl))
            doEvents(&eventStructure);
        else
        {
            if(eventStructure.what == nullEvent)
            {
                doldle();
                doSynchroniseFiles();
            }
        }
    }
}

// ***** doGetSleepTime

UInt32 doGetSleepTime(void)
{
    WindowRef windowRef;
    UInt32 sleepTime;
    TXNObject txnObject = NULL;

    windowRef = FrontWindow();

    if(isApplicationWindow(windowRef,&txnObject))
        sleepTime = TXNGetSleepTicks(txnObject);
    else
        sleepTime = GetCaretTime();

    return sleepTime;
}

// ***** doldle

void doldle(void)
{
    WindowRef windowRef;
    TXNObject txnObject = NULL;

    windowRef = FrontWindow();
    if(isApplicationWindow(windowRef,&txnObject))
    {
        if(TXNGetChangeCount(txnObject))
            SetWindowModified(windowRef,true);
    }
}

// ***** doEvents

void doEvents(EventRecord *eventStrucPtr)
{
    WindowRef windowRef;
    TXNObject txnObject = NULL;

    switch(eventStrucPtr->what)
    {

```

```

case kHighLevelEvent:
    AEResponseAppleEvent(eventStrucPtr);
    break;

case mouseDown:
    doMouseDown(eventStrucPtr);
    break;

case keyDown:
    if(eventStrucPtr->modifiers & cmdKey)
    {
        doAdjustAndPrepareMenus();
        doMenuChoice(MenuEvent(eventStrucPtr));
    }
    break;

case updateEvt:
    doUpdate(eventStrucPtr);
    break;

case activateEvt:
    doActivate(eventStrucPtr);
    break;

case osEvt:
    switch((eventStrucPtr->message >> 24) & 0x000000FF)
    {
        case suspendResumeMessage:
            if(eventStrucPtr->message & resumeFlag)
                SetThemeCursor(kThemeArrowCursor);
            break;

        case mouseMovedMessage:
            windowRef = FrontWindow();
            if(isApplicationWindow(windowRef,&txnObject))
                TXNAdjustCursor(txnObject,gCursorRgnHdl);
    }
    break;
}
}

// ***** doMouseDown

void doMouseDown(EventRecord *eventStrucPtr)
{
    WindowRef    windowRef;
    WindowPartCode partCode;
    OSStatus     osStatus = noErr;
    TXNObject    txnObject = NULL;
    Boolean      handled = false;
    SInt32       itemSelected;

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

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

        case inContent:
            if(windowRef != FrontWindow())
                SelectWindow(windowRef);
            else
            {
                if(isApplicationWindow(windowRef,&txnObject))
                    TXNClick(txnObject,eventStrucPtr);
            }
            break;

        case inGoAway:
            if(TrackGoAway(windowRef,eventStrucPtr->where))
                doCloseCommand(kNavSaveChangesClosingDocument);
            break;

        case inProxyIcon:
            osStatus = TrackWindowProxyDrag(windowRef,eventStrucPtr->where);
            if(osStatus == errUserWantsToDragWindow)

```

```

    handled = false;
else if(osStatus == noErr)
    handled = true;

case inDrag:
if(!handled)
{
    if(IsWindowPathSelectClick(windowRef,eventStrucPtr))
    {
        if(WindowPathSelect(windowRef,NULL,&itemSelected) == noErr)
        {
            if(LoWord(itemSelected) > 1)
                doBringFinderToFront();
        }

        handled = true;
    }
}
if(!handled)
    DragWindow(windowRef,eventStrucPtr->where,NULL);

if(isApplicationWindow(windowRef,&txnObject))
    TXNAdjustCursor(txnObject,gCursorRgnHdl);

break;

case inGrow:
if(isApplicationWindow(windowRef,&txnObject))
{
    TXNGrowWindow(txnObject,eventStrucPtr);
    TXNAdjustCursor(txnObject,gCursorRgnHdl);
}
break;

case inZoomIn:
case inZoomOut:
if(TrackBox(windowRef,eventStrucPtr->where,partCode))
{
    if(isApplicationWindow(windowRef,&txnObject))
    {
        TXNZoomWindow(txnObject,partCode);
        TXNAdjustCursor(txnObject,gCursorRgnHdl);
    }
}
break;
}
}

// ***** doBringFinderToFront

void doBringFinderToFront(void)
{
    ProcessSerialNumber finderProcess;

    if(doFindProcess('MACS','FNDR",&finderProcess) == noErr)
        SetFrontProcess(&finderProcess);
    else
        doErrorAlert(eCantFindFinderProcess);
}

// ***** doFindProcess

OSStatus doFindProcess(OSType creator,OSType type,ProcessSerialNumber *outProcSerNo)
{
    ProcessSerialNumber procSerialNo;
    ProcessInfoRec    procInfoStruc;
    OSStatus          osStatus = noErr;

    procSerialNo.highLongOfPSN = 0;
    procSerialNo.lowLongOfPSN = kNoProcess;

    procInfoStruc.processInfoLength = sizeof(ProcessInfoRec);
    procInfoStruc.processName       = NULL;
    procInfoStruc.processAppSpec    = NULL;
    procInfoStruc.processLocation   = NULL;

    while(true)
    {
        osStatus = GetNextProcess(&procSerialNo);

```

```

    if(osStatus != noErr)
        break;

    osStatus = GetProcessInformation(&procSerialNo,&procInfoStruc);
    if(osStatus != noErr)
        break;
    if((procInfoStruc.processSignature == creator) && (procInfoStruc.processType == type))
        break;
}

*outProcSerNo = procSerialNo;

return osStatus;
}

// ***** doActivate

void doActivate(EventRecord *eventStrucPtr)
{
    WindowRef windowRef;
    TXNObject txnObject = NULL;
    Boolean becomingActive;
    TXNFrameID txnFrameID = 0;

    windowRef = (WindowRef) eventStrucPtr->message;

    if(isApplicationWindow(windowRef,&txnObject))
    {
        becomingActive = ((eventStrucPtr->modifiers & activeFlag) == activeFlag);
        GetWindowProperty(windowRef,kFileCreator,'tFRM',sizeof(TXNFrameID),NULL,&txnFrameID);

        if(becomingActive)
            TXNActivate(txnObject,txnFrameID,becomingActive);
        else
            TXNActivate(txnObject,txnFrameID,becomingActive);

        TXNFocus(txnObject,becomingActive);
    }
}

// ***** doUpdate

void doUpdate(EventRecord *eventStrucPtr)
{
    WindowRef windowRef;
    GrafPtr oldPort;
    TXNObject txnObject = NULL;

    windowRef = (WindowRef) eventStrucPtr->message;

    GetPort(&oldPort);
    SetPortWindowPort(windowRef);

    if(isApplicationWindow(windowRef,&txnObject))
        TXNUpdate(txnObject);

    SetPort(oldPort);
}

// ***** isApplicationWindow

Boolean isApplicationWindow(WindowRef windowRef,TXNObject *txnObject)
{
    OSStatus osStatus = noErr;

    osStatus = GetWindowProperty(windowRef,kFileCreator,'tOBJ',sizeof(TXNObject),NULL,
        txnObject);

    return (windowRef != NULL) && (GetWindowKind(windowRef) == kApplicationWindowKind);
}

// ***** doAboutDialog

void doAboutDialog(void)
{
    DialogRef dialogRef;
    SInt16 itemHit;

    dialogRef = GetNewDialog(rAboutDialog,NULL,(WindowRef) -1);
}

```

```

ModalDialog(NULL,&itemHit);
DisposeDialog(dialogRef);
}

// ***** doSynchroniseFiles

void doSynchroniseFiles(void)
{
    UInt32    currentTicks;
    WindowRef windowRef;
    static UInt32 nextSynchTicks = 0;
    OSStatus  hasNoAliasHdl = noErr;
    Boolean    aliasChanged;
    AliasHandle aliasHdl = NULL;
    FSSpec     newFSSpec;
    OSStatus   osStatus = noErr;
    SInt16     trashVRefNum;
    SInt32     trashDirID;
    TXNObject  txnObject = NULL;

    currentTicks = TickCount();
    windowRef = FrontWindow();

    if(currentTicks > nextSynchTicks)
    {
        while(windowRef != NULL)
        {
            hasNoAliasHdl = GetWindowProperty(windowRef,kFileCreator,'tALH',sizeof(AliasHandle),
                NULL,&aliasHdl);
            if(hasNoAliasHdl)
                break;

            aliasChanged = false;
            ResolveAlias(NULL,aliasHdl,&newFSSpec,&aliasChanged);
            if(aliasChanged)
            {
                SetWindowProperty(windowRef,kFileCreator,'FiSp',sizeof(FSSpec),&newFSSpec);
                SetWTitle(windowRef,newFSSpec.name);
            }

            osStatus = FindFolder(kUserDomain,kTrashFolderType,kDontCreateFolder,
                &trashVRefNum,&trashDirID);

            if(osStatus == noErr)
            {
                do
                {
                    if(newFSSpec.parID == fsRtParID)
                        break;

                    if((newFSSpec.vRefNum == trashVRefNum) && (newFSSpec.parID == trashDirID))
                    {
                        GetWindowProperty(windowRef,kFileCreator,'tOBJ',sizeof(TXNObject),NULL,
                            &txnObject);
                        TXNDeleteObject(txnObject);
                        DisposeWindow(windowRef);
                        gCurrentNumberOfWindows --;
                        break;
                    }
                } while(FSMakeFSSpec(newFSSpec.vRefNum,newFSSpec.parID,"\\p",&newFSSpec) == noErr);
            }

            windowRef = GetNextWindow(windowRef);
        }

        nextSynchTicks = currentTicks + 15;
    }
}

// ***** openAppEventHandler

OSStatus openAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefCon)
{
    OSStatus osStatus = noErr;

    osStatus = doHasGotRequiredParams(appEvent);
    if(osStatus == noErr)
        osStatus = doNewCommand();
}

```



```

    return osStatus;
}

// ***** reopenAppEventHandler

OSStatus reopenAppEventHandler(AppleEvent *appEvent, AppleEvent *reply,
                               SInt32 handlerRefCon)
{
    OSStatus osStatus = noErr;

    osStatus = doHasGotRequiredParams(appEvent);
    if(osStatus == noErr)
        if(!FrontWindow())
            osStatus = doNewCommand();

    return osStatus;
}

// ***** openAndPrintDocsEventHandler

OSStatus openAndPrintDocsEventHandler(AppleEvent *appEvent, AppleEvent *reply,
                                       SInt32 handlerRefcon)
{
    FSSpec fileSpec;
    AEDescList docList;
    OSStatus osStatus, ignoreErr;
    SInt32 index, numberOfItems;
    Size actualSize;
    AEKeyword keyWord;
    DescType returnedType;
    FInfo fileInfo;
    TXNObject txnObject;

    osStatus = AEGgetParamDesc(appEvent, keyDirectObject, typeAEList, &docList);

    if(osStatus == noErr)
    {
        osStatus = doHasGotRequiredParams(appEvent);
        if(osStatus == noErr)
        {
            osStatus = AECcountItems(&docList, &numberOfItems);
            if(osStatus == noErr)
            {
                for(index=1; index<=numberOfItems; index++)
                {
                    osStatus = AEGgetNthPtr(&docList, index, typeFSSpec, &keyWord, &returnedType,
                                             &fileSpec, sizeof(fileSpec), &actualSize);
                    if(osStatus == noErr)
                    {
                        osStatus = FSpGetFInfo(&fileSpec, &fileInfo);
                        if(osStatus == noErr)
                        {
                            if(osStatus = doOpenFile(fileSpec, fileInfo.fdType))
                                doErrorAlert(osStatus);

                            if(osStatus == noErr && handlerRefcon == kPrint)
                            {
                                if(isApplicationWindow(FrontWindow(), &txnObject))
                                {
                                    if(osStatus = TXNPrint(txnObject))
                                        doErrorAlert(osStatus);

                                    if(osStatus = doCloseCommand(kNavSaveChangesOther))
                                        doErrorAlert(osStatus);
                                }
                            }
                        }
                    }
                }
            }
            else
                doErrorAlert(osStatus);
        }
    }
    else
        doErrorAlert(osStatus);

    ignoreErr = AEDisposeDesc(&docList);
}
else

```

```

doErrorAlert(osStatus);

return osStatus;
}

// ***** quitAppEventHandler

OSStatus quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
    OSStatus osStatus = noErr;

    osStatus = doHasGotRequiredParams(appEvent);
    if(osStatus == noErr)
    {
        while(FrontWindow())
        {
            osStatus = doCloseCommand(kNavSaveChangesQuittingApplication);

            if(osStatus != noErr && osStatus != kNavAskSaveChangesCancel)
                doErrorAlert(osStatus);
            if(osStatus == kNavAskSaveChangesCancel)
                return noErr;
        }
    }

    gDone = true;

    return osStatus;
}

// ***** doHasGotRequiredParams

OSStatus doHasGotRequiredParams(AppleEvent *appEvent)
{
    DescType returnedType;
    Size actualSize;
    OSStatus osStatus = noErr;

    osStatus = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,
        NULL,0,&actualSize);
    if(osStatus == errAEDescNotFound)
        osStatus = noErr;
    else if(osStatus == noErr)
        osStatus = errAEParamMissed;

    return osStatus;
}

// ***** doErrorAlert

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

    if(errorCode == kATSUFontsMatched)
        return;

    if(errorCode == eInstallHandler)
        GetIndString(errorString,rErrorStrings,1);
    else if(errorCode == eMaxWindows)
        GetIndString(errorString,rErrorStrings,2);
    else if(errorCode == eCantFindFinderProcess)
        GetIndString(errorString,rErrorStrings,3);
    else
    {
        GetIndString(errorString,rErrorStrings,4);
        NumToString((SInt32) errorCode,theString);
        doConcatPStrings(errorString,theString);
    }

    if(errorCode != memFullErr)
        StandardAlert(kAlertCautionAlert,errorString,NULL,NULL,&itemHit);
    else
    {
        StandardAlert(kAlertStopAlert,errorString,NULL,NULL,&itemHit);
        ExitToShell();
    }
}

```

```

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

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

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

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

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

// *****
// MLTEMenus.c
// *****

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

#include "MLTETextEditor.h"

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

RGBColor          gCurrentColourPickerColour = { 0x0000,0x0000,0x0000 };
extern SInt16      gCurrentNumberOfWindows;
extern TXNFontMenuObject gTXNFontMenuObject;
extern Boolean     gDone;

// ***** doEnableDisableMenus
void doEnableDisableMenus(Boolean enableMenus)
{
    if(enableMenus)
    {
        EnableMenuItem(GetMenuRef(mEdit),0);
        EnableMenuItem(GetMenuRef(mFont),0);
        EnableMenuItem(GetMenuRef(mSize),0);
        EnableMenuItem(GetMenuRef(mStyle),0);
        EnableMenuItem(GetMenuRef(mColour),0);
        EnableMenuItem(GetMenuRef(mJustification),0);
        EnableMenuItem(GetMenuRef(mWindow),0);
    }
    else
    {
        DisableMenuItem(GetMenuRef(mEdit),0);
        DisableMenuItem(GetMenuRef(mFont),0);
        DisableMenuItem(GetMenuRef(mSize),0);
        DisableMenuItem(GetMenuRef(mStyle),0);
        DisableMenuItem(GetMenuRef(mColour),0);
        DisableMenuItem(GetMenuRef(mJustification),0);
        DisableMenuItem(GetMenuRef(mWindow),0);
    }
}

// ***** doAdjustAndPrepareMenus
void doAdjustAndPrepareMenus(void)
{
    WindowRef windowRef;

    windowRef = FrontWindow();
}

```

```

doAdjustFileMenu(GetMenuRef(mFile),windowRef);
doAdjustEditMenu(GetMenuRef(mEdit),windowRef);
doPrepareFontMenu(windowRef);
doPrepareSizeMenu(GetMenuRef(mSize),windowRef);
doPrepareStyleMenu(GetMenuRef(mStyle),windowRef);
doPrepareColourMenu(GetMenuRef(mColour),windowRef);
doPrepareJustificationMenu(GetMenuRef(mJustification),windowRef);

DrawMenuBar();
}

// ***** doAdjustFileMenu

void doAdjustFileMenu(MenuRef menuRef,WindowRef windowRef)
{
    TXNObject txnObject = NULL;

    if(gCurrentNumberOfWindows <= kMaxWindows)
    {
        EnableMenuItem(menuRef,iNew);
        EnableMenuItem(menuRef,iOpen);
    }
    else
    {
        DisableMenuItem(menuRef,iNew);
        DisableMenuItem(menuRef,iOpen);
    }

    if(isApplicationWindow(windowRef,&txnObject))
    {
        EnableMenuItem(menuRef,iClose);

        if(TXNGetChangeCount(txnObject))
        {
            EnableMenuItem(menuRef,iSave);
            EnableMenuItem(menuRef,iRevert);
        }
        else
        {
            DisableMenuItem(menuRef,iSave);
            DisableMenuItem(menuRef,iRevert);
        }

        if(TXNDataSize(txnObject))
        {
            EnableMenuItem(menuRef,iSaveAs);
            EnableMenuItem(menuRef,iPageSetup);
            EnableMenuItem(menuRef,iPrint);
        }
        else
        {
            DisableMenuItem(menuRef,iSaveAs);
            DisableMenuItem(menuRef,iPageSetup);
            DisableMenuItem(menuRef,iPrint);
        }
    }
    else
    {
        DisableMenuItem(menuRef,iClose);
        DisableMenuItem(menuRef,iSave);
        DisableMenuItem(menuRef,iSaveAs);
        DisableMenuItem(menuRef,iRevert);
        DisableMenuItem(menuRef,iPageSetup);
        DisableMenuItem(menuRef,iPrint);
    }
}

// ***** doAdjustEditMenu

void doAdjustEditMenu(MenuRef menuRef,WindowRef windowRef)
{
    TXNObject txnObject = NULL;
    SInt16 menuItem;
    Str255 itemText;
    TXNActionKey actionKey;

    if(isApplicationWindow(windowRef,&txnObject))
    {

```

```

//
..... disable all items

for(menuItem = iUndo;menuItem <= iSelectAll;menuItem++)
    DisableMenuItem(menuRef,menuItem);

// ..... undo and redo default - can't undo, can't
redo

GetIndString(itemText,130,1);
SetMenuItemText(menuRef,iUndo,itemText);
GetIndString(itemText,130,2);
SetMenuItemText(menuRef,iRedo,itemText);

// ..... if undoable, enable undo item and set item
text

if(TXNCanUndo(txnObject,&actionKey))
{
    EnableMenuItem(menuRef,iUndo);

    if((actionKey < kTXNTypingAction) || (actionKey > kTXNMoveAction))
        actionKey = -1;

    GetIndString(itemText,130,2 * actionKey + 5);
    SetMenuItemText(menuRef,iUndo,itemText);
}

// ..... if redoable, enable redo item and set item
text

if(TXNCanRedo(txnObject,&actionKey))
{
    EnableMenuItem(menuRef,iRedo);

    if((actionKey < kTXNTypingAction) || (actionKey > kTXNMoveAction))
        actionKey = -1;

    GetIndString(itemText,130,2 * actionKey + 6);
    SetMenuItemText(menuRef,iRedo,itemText);
}

// ..... if there is a selection, enable cut, copy, and clear

if(!TXNIsSelectionEmpty(txnObject))
{
    EnableMenuItem(menuRef,iCut);
    EnableMenuItem(menuRef,iCopy);
    EnableMenuItem(menuRef,iClear);
}

// ..... if scrap is
pastable, enable paste

if(TXNIsScrapPastable())
    EnableMenuItem(menuRef,iPaste);

// ..... if any characters in TXNObject, enable select
all

if(TXNDataSize(txnObject))
    EnableMenuItem(menuRef,iSelectAll);
}
}

// ***** doPrepareFontMenu

void doPrepareFontMenu(WindowRef windowRef)
{
    TXNObject txnObject = NULL;

    if(isApplicationWindow(windowRef,&txnObject))
        TXNPrepareFontMenu(txnObject,gTXNFontMenuObject);
}

// ***** doPrepareSizeMenu

void doPrepareSizeMenu(MenuRef menuRef,WindowRef windowRef)

```

```

{
TXNObject      txnObject = NULL;
static Fixed   itemSizes[8] = { 0x00090000,0x000A0000,0x000B0000,0x000C0000,
                                0x000E0000,0x00120000,0x00180000,0x00240000 };
TXNContinuousFlags txnContinuousFlags = 0;
TXNTypeAttributes txnTypeAttributes;
OSStatus        osStatus = noErr;
SInt16          menuItem;

if(isApplicationWindow(windowRef,&txnObject))
{
    txnTypeAttributes.tag      = kTXNQDFontSizeAttribute;
    txnTypeAttributes.size     = kTXNQDFontSizeAttributeSize;
    txnTypeAttributes.data.dataValue = 0;

    osStatus = TXNGetContinuousTypeAttributes(txnObject,&txnContinuousFlags,1,
                                              &txnTypeAttributes);
    if(osStatus == noErr)
    {
        for(menuItem = 1;menuItem < 8;menuItem ++)
        {
            CheckMenuItem(menuRef,menuItem,(txnContinuousFlags & kTXNSizeContinuousMask) &&
                          (txnTypeAttributes.data.dataValue == itemSizes[menuItem - 1]));
        }
    }
}
}

// ***** doPrepareStyleMenu

void doPrepareStyleMenu(MenuRef menuRef,WindowRef windowRef)
{
TXNObject      txnObject = NULL;
TXNContinuousFlags txnContinuousFlags = 0;
TXNTypeAttributes txnTypeAttributes;
OSStatus        osStatus = noErr;
SInt16          menuItem;

if(isApplicationWindow(windowRef,&txnObject))
{
    txnTypeAttributes.tag      = kTXNQDFontStyleAttribute;
    txnTypeAttributes.size     = kTXNQDFontStyleAttributeSize;
    txnTypeAttributes.data.dataValue = 0;

    osStatus = TXNGetContinuousTypeAttributes(txnObject,&txnContinuousFlags,1,
                                              &txnTypeAttributes);
    if(osStatus == noErr)
    {
        CheckMenuItem(menuRef,iPlain,(txnContinuousFlags & kTXNStyleContinuousMask) &&
                      (txnTypeAttributes.data.dataValue == normal));

        for(menuItem = iBold;menuItem <= iUnderline;menuItem ++)
        {
            CheckMenuItem(menuRef,menuItem,(txnContinuousFlags & kTXNStyleContinuousMask) &&
                          (txnTypeAttributes.data.dataValue & (1 << (menuItem - iBold))));
        }
    }
}
}

// ***** doPrepareColourMenu

void doPrepareColourMenu(MenuRef menuRef,WindowRef windowRef)
{
TXNObject      txnObject = NULL;
TXNContinuousFlags txnContinuousFlags = 0;
TXNTypeAttributes txnTypeAttributes;
RGBColor       attributesColour;
OSStatus        osStatus = noErr;
SInt16          menuItem;
RGBColor       itemColours[4] = { { 0xFFFF,0x0000,0x0000 },{ 0x0000,0x8888,0x0000 },
                                  { 0x0000,0x0000,0xFFFF },{ 0x0000,0x0000,0x0000 } };

if(isApplicationWindow(windowRef,&txnObject))
{
    txnTypeAttributes.tag      = kTXNQDFontColorAttribute;
    txnTypeAttributes.size     = kTXNQDFontColorAttributeSize;
    txnTypeAttributes.data.dataPtr = &attributesColour;
}
}

```

```

osStatus = TXNGetContinuousTypeAttributes(txnObject,&txnContinuousFlags,1,
&txnTypeAttributes);
if(osStatus == noErr)
{
for(menuItem = 1;menuItem < 5;menuItem ++)
{
CheckMenuItem(menuRef,menuItem,(txnContinuousFlags & kTXNColorContinuousMask) &&
(isEqualRGB(&attributesColour,&itemColours[menuItem - 1])));
}
}
}
}

// ***** isEqualRGB

Boolean isEqualRGB(RGBColor *attributesColour,RGBColor *itemColour)
{
return (attributesColour->red == itemColour->red &&
attributesColour->green == itemColour->green &&
attributesColour->blue == itemColour->blue);
}

// ***** doPrepareJustificationMenu

void doPrepareJustificationMenu (MenuRef menuRef,WindowRef windowRef)
{
TXNObject txnObject = NULL;
static UInt32 itemJustifications[6] = { kTXNFlushDefault,kTXNFlushLeft,kTXNFlushRight,
kTXNCenter,kTXNFullJust,kTXNForceFullJust};
TXNControlTag txnControlTag[1];
TXNControlData txnControlData[1];
SInt16 menuItem;
OSStatus osStatus = noErr;

if(isApplicationWindow(windowRef,&txnObject))
{
txnControlTag[0] = kTXNJustificationTag ;
txnControlData[0].uValue = 0;

osStatus = TXNGetTXNObjectControls(txnObject,1,txnControlTag,txnControlData);

if(osStatus == noErr)
{
for(menuItem = iDefault;menuItem <= iForceFull;menuItem ++ )
CheckMenuItem(menuRef,menuItem,(txnControlData[0].uValue ==
itemJustifications[menuItem - 1]));
}
}
}

// ***** doMenuChoice

void doMenuChoice(SInt32 menuChoice)
{
MenuID menuID;
MenuItemIndex menuItem;
OSStatus osStatus = noErr;
WindowRef windowRef;
TXNObject txnObject = NULL;

windowRef = FrontWindow();

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

if(menuID == 0)
return;

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

case mFile:
doFileMenuChoice(menuItem,windowRef);
break;
}
}

```

```

case mEdit:
    doEditMenuChoice(menuItem,windowRef);
    break;

case mFont:
    doFontMenuChoice(menuID,menuItem,windowRef);
    break;

case mSize:
    doSizeMenuChoice(menuItem,windowRef);
    break;

case mStyle:
    doStyleMenuChoice(menuItem,windowRef);
    break;

case mColour:
    doColourMenuChoice(menuItem,windowRef);
    break;

case mJustification:
    doJustificationMenuChoice(menuItem,windowRef);
    break;

default:
    if(menuID >= mFirstHierarchical)
        doFontMenuChoice(menuID,menuItem,windowRef);
    break;
}

HiliteMenu(0);
}

// ***** doFileMenuChoice

void doFileMenuChoice(MenuBarItem menuBarItem,WindowRef windowRef)
{
    TXNObject txnObject = NULL;
    OSStatus osStatus = noErr;

    switch(menuBarItem)
    {
        case iNew:
            if(osStatus = doNewCommand())
                doErrorAlert(osStatus);
            break;

        case iOpen:
            if(osStatus = doOpenCommand())
                doErrorAlert(osStatus);
            break;

        case iClose:
            if((osStatus = doCloseCommand(kNavSaveChangesClosingDocument)) &&
                osStatus != kNavAskSaveChangesCancel)
                doErrorAlert(osStatus);
            break;

        case iSave:
            if(osStatus = doSaveCommand())
                doErrorAlert(osStatus);
            break;

        case iSaveAs:
            if(osStatus = doSaveAsCommand())
                doErrorAlert(osStatus);
            break;

        case iRevert:
            if(osStatus = doRevertCommand())
                doErrorAlert(osStatus);
            break;

        case iPageSetup:
            if(isApplicationWindow(windowRef,&txnObject))
            {
                osStatus = TXNPageSetup(txnObject);
                if(osStatus != userCanceledErr && osStatus != noErr)
                    doErrorAlert(osStatus);
            }
    }
}

```



```

    }
    break;

case iPrint:
    if(isApplicationWindow(windowRef,&txnObject))
    {
        osStatus = TXNPrint(txnObject);
        if(osStatus != userCanceledErr && osStatus != noErr)
            doErrorAlert(osStatus);
    }
    break;

case iQuit:
    if((osStatus = doQuitCommand(kNavSaveChangesQuittingApplication)) &&
        osStatus != kNavAskSaveChangesCancel)
        doErrorAlert(osStatus);

    if(osStatus != kNavAskSaveChangesCancel)
    {
        if(gTXNFontMenuObject != NULL)
        {
            if(osStatus = TXNDisposeFontMenuObject(gTXNFontMenuObject))
                doErrorAlert(osStatus);
        }

        gTXNFontMenuObject = NULL;

        TXNTerminateTextension();
        gDone = true;
    }
    break;
}
}

// ***** doEditMenuChoice

void doEditMenuChoice(MenuBarItem menuBarItem,WindowRef windowRef)
{
    TXNObject txnObject = NULL;
    OSStatus osStatus = noErr;

    if(isApplicationWindow(windowRef,&txnObject))
    {
        switch(menuBarItem)
        {
            case iUndo:
                TXNUndo(txnObject);
                break;

            case iRedo:
                TXNRedo(txnObject);
                break;

            case iCut:
                if((osStatus = TXNCut(txnObject)) == noErr)
                    TXNConvertToPublicScrap();
                else
                    doErrorAlert(osStatus);
                break;

            case iCopy:
                if((osStatus = TXNCopy(txnObject)) == noErr)
                    TXNConvertToPublicScrap();
                else
                    doErrorAlert(osStatus);
                break;

            case iPaste:
                if(osStatus = TXNPaste(txnObject))
                    doErrorAlert(osStatus);
                break;

            case iClear:
                if(osStatus = TXNClear(txnObject))
                    doErrorAlert(osStatus);
                break;

            case iSelectAll:
                TXNSelectAll(txnObject);
        }
    }
}

```

```

        break;
    }
}
}

// ***** doFontMenuChoice

void doFontMenuChoice(MenuID menuID,MenuItemIndex menuItem,WindowRef windowRef)
{
    TXNObject txnObject = NULL;
    OSStatus osStatus = noErr;

    if(isApplicationWindow(windowRef,&txnObject))
    {
        if(gTXNFontMenuObject != NULL)
        {
            if(osStatus = TXNDoFontMenuSelection(txnObject,gTXNFontMenuObject,menuID,menuItem))
                doErrorAlert(osStatus);
        }
    }
}

// ***** doSizeMenuChoice

void doSizeMenuChoice(MenuItemIndex menuItem,WindowRef windowRef)
{
    TXNObject    txnObject = NULL;
    static Fixed  itemSizes[8] = { 0x00090000,0x000A0000,0x000B0000,0x000C0000,
                                   0x000E0000,0x00120000,0x00180000,0x00240000 };
    Fixed        sizeToSet;
    TXNTypeAttributes txnTypeAttributes;
    OSStatus     osStatus = noErr;

    if(isApplicationWindow(windowRef,&txnObject))
    {
        sizeToSet = itemSizes[menuItem - 1];

        txnTypeAttributes.tag      = kTXNQDFontSizeAttribute;
        txnTypeAttributes.size     = kTXNFontSizeAttributeSize;
        txnTypeAttributes.data.dataValue = sizeToSet;

        if(TXNSetTypeAttributes(txnObject,1,&txnTypeAttributes,kTXNUseCurrentSelection,
                                kTXNUseCurrentSelection))
            doErrorAlert(osStatus);
    }
}

// ***** doStyleMenuChoice

void doStyleMenuChoice(MenuItemIndex menuItem,WindowRef windowRef)
{
    TXNObject    txnObject = NULL;
    static Style  itemStyles[5] = { normal,0,bold,italic,underline };
    Style        styleToSet;
    TXNTypeAttributes txnTypeAttributes;
    OSStatus     osStatus = noErr;

    if(isApplicationWindow(windowRef,&txnObject))
    {
        styleToSet = itemStyles[menuItem - 1];

        txnTypeAttributes.tag      = kTXNQDFontStyleAttribute;
        txnTypeAttributes.size     = kTXNQDFontStyleAttributeSize;
        txnTypeAttributes.data.dataValue = styleToSet;

        if(TXNSetTypeAttributes(txnObject,1,&txnTypeAttributes,kTXNUseCurrentSelection,
                                kTXNUseCurrentSelection))
            doErrorAlert(osStatus);
    }
}

// ***** doColourMenuChoice

void doColourMenuChoice(MenuItemIndex menuItem,WindowRef windowRef)
{
    TXNObject    txnObject = NULL;
    Point        where;
    Boolean       colorPickerButton;
    Str255       prompt = "\pPick a text colour";
}

```

```

RGBColor    colourToSet;
RGBColor    itemColours[4] = { { 0xFFFF,0x0000,0x0000 },{ 0x0000,0x8888,0x0000 },
                               { 0x0000,0x0000,0xFFFF },{ 0x0000,0x0000,0x0000 } };
TXNTypeAttributes txnTypeAttributes;
OSStatus     osStatus = noErr;

if(isApplicationWindow(windowRef,&txnObject))
{
    if(menuItem == iColourPicker)
    {
        where.v = where.h = 0;
        colorPickerButton = GetColor(where,prompt,&gCurrentColourPickerColour,&colourToSet);
        if(colorPickerButton)
            gCurrentColourPickerColour = colourToSet;
        else
            return;
    }
    else
        colourToSet = itemColours[menuItem - 1];

    txnTypeAttributes.tag = kTXNQDFontColorAttribute;
    txnTypeAttributes.size = kTXNQDFontColorAttributeSize;
    txnTypeAttributes.data.dataPtr = &colourToSet;

    if(TXNSetTypeAttributes(txnObject,1,&txnTypeAttributes,kTXNUseCurrentSelection,
                           kTXNUseCurrentSelection))
        doErrorAlert(osStatus);
}
}

// ***** doJustificationMenuChoice

void doJustificationMenuChoice(MenuItemIndex menuItem,WindowRef windowRef)
{
    TXNObject    txnObject = NULL;
    static UInt32 itemJustifications[6] = { kTXNFlushDefault,kTXNFlushLeft,kTXNFlushRight,
                                           kTXNCenter,kTXNFullJust,kTXNForceFullJust };
    OSStatus     osStatus = noErr;
    UInt32       justificationToSet;
    TXNControlTag txnControlTag[1];
    TXNControlData txnControlData[1];

    if(isApplicationWindow(windowRef,&txnObject))
    {
        justificationToSet = itemJustifications[menuItem - 1];

        txnControlTag[0] = kTXNJustificationTag;

        osStatus = TXNGetTXNObjectControls(txnObject,1,txnControlTag,txnControlData);

        if(txnControlData[0].uValue != justificationToSet)
        {
            txnControlData[0].uValue = justificationToSet;
            osStatus = TXNSetTXNObjectControls(txnObject,false,1,txnControlTag,txnControlData);
            if(osStatus != noErr)
                doErrorAlert(osStatus);
        }
    }
}

// *****
// MLTNewOpenCloseSave.c
// *****

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

#include "MLTETextEditor.h"

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

SInt16 gCurrentNumberOfWindows = 0;
SInt16 gUntitledWindowNumber = 0;

extern Boolean gRunningOnX = false;
extern SInt16 gAppResFileRefNum;

```

```

// ***** doNewCommand
OSStatus doNewCommand(void)
{
    OSStatus osStatus = noErr;
    WindowRef windowRef;

    if(gCurrentNumberOfWindows == kMaxWindows)
        return eMaxWindows;

    osStatus = doNewDocWindow(&windowRef,NULL,kTXNTextensionFile);

    if(osStatus == noErr)
        SetWindowProxyCreatorAndType(windowRef,kFileCreator,kTXNTextensionFile,kUserDomain);

    return osStatus;
}

// ***** doOpenCommand
OSStatus doOpenCommand(void)
{
    OSStatus osStatus = noErr;
    NavDialogOptions dialogOptions;
    NavTypeListHandle fileTypeListHdl = NULL;
    NavEventUPP navEventFunctionUPP;
    NavReplyRecord navReplyStruc;
    SInt32 count, index;
    AEKeyword theKeyword;
    DescType actualType;
    FSSpec fileSpec;
    Size actualSize;
    FInfo fileInfo;
    OSType fileType;

    osStatus = NavGetDefaultDialogOptions(&dialogOptions);

    if(osStatus == noErr)
    {
        GetIndString(dialogOptions.clientName,rMiscellaneousStrings,sApplicationName);
        fileTypeListHdl = (NavTypeListHandle) GetResource('open',rOpenResource);

        navEventFunctionUPP = NewNavEventUPP((NavEventProcPtr) navEventFunction);

        osStatus = NavGetFile(NULL,&navReplyStruc,&dialogOptions,navEventFunctionUPP,
            NULL,NULL,fileTypeListHdl,NULL);

        DisposeNavEventUPP(navEventFunctionUPP);

        if(osStatus == noErr && navReplyStruc.validRecord)
        {
            osStatus = AECntItems(&(navReplyStruc.selection),&count);
            if(osStatus == noErr)
            {
                for(index=1;index<=count;index++)
                {
                    osStatus = AEGetNthPtr(&(navReplyStruc.selection),index,typeFSS,&theKeyword,
                        &actualType,&fileSpec,sizeof(fileSpec),&actualSize);

                    if((osStatus = FSpGetFInfo(&fileSpec,&fileInfo)) == noErr)
                    {
                        fileType = fileInfo.fdType;
                        osStatus = doOpenFile(fileSpec,fileType);
                    }
                }
            }

            osStatus = NavDisposeReply(&navReplyStruc);
        }

        if(fileTypeListHdl != NULL)
            ReleaseResource((Handle) fileTypeListHdl);
    }

    if(osStatus == userCanceledErr)
        osStatus = noErr;

    return osStatus;
}

```

```

}

// ***** doCloseCommand

OSStatus doCloseCommand(NavAskSaveChangesAction action)
{
    WindowRef      windowRef;
    TXNObject      txnObject = NULL;
    OSStatus       osStatus = noErr;
    NavDialogOptions dialogOptions;
    NavAskSaveChangesResult reply = 0;
    NavEventUPP    navEventFunctionUPP;
    Str255         fileName;

    osStatus = NavGetDefaultDialogOptions(&dialogOptions);

    if(osStatus == noErr)
    {
        windowRef = FrontWindow();
        if(isApplicationWindow(windowRef,&txnObject))
        {
            if(TXNGetChangeCount(txnObject))
            {
                GetWTitle(windowRef,fileName);
                BlockMoveData(fileName,dialogOptions.savedFileName,fileName[0] + 1);
                GetIndString(dialogOptions.clientName,rMiscellaneousStrings,sApplicationName);

                navEventFunctionUPP = NewNavEventUPP((NavEventProcPtr) navEventFunction);

                osStatus = NavAskSaveChanges(&dialogOptions,action,&reply,navEventFunctionUPP,0);

                DisposeNavEventUPP(navEventFunctionUPP);

                if(osStatus == noErr)
                {
                    switch(reply)
                    {
                        case kNavAskSaveChangesSave:
                            if((osStatus = doSaveCommand()) == noErr)
                                doCloseWindow(windowRef,txnObject);
                            break;

                        case kNavAskSaveChangesDontSave:
                            doCloseWindow(windowRef,txnObject);
                            break;

                        case kNavAskSaveChangesCancel:
                            osStatus = kNavAskSaveChangesCancel;
                            break;
                    }
                }
            }
            else
            {
                doCloseWindow(windowRef,txnObject);
            }
        }
    }

    return osStatus;
}

// ***** doSaveCommand

OSStatus doSaveCommand(void)
{
    WindowRef windowRef;
    OSStatus hasNoFileSpec;
    OSStatus osStatus = noErr;
    FSSpec fileSpec;

    windowRef = FrontWindow();

    hasNoFileSpec = GetWindowProperty(windowRef,kFileCreator,'FiSp',sizeof(FSSpec),NULL,
                                     &fileSpec);
    if(hasNoFileSpec)
        osStatus = doSaveAsCommand();
    else
        osStatus = doWriteFile(windowRef,false);
}

```

```

if(osStatus == noErr)
    SetWindowModified(windowRef,false);

return osStatus;
}

// ***** doSaveAsCommand

OSStatus doSaveAsCommand(void)
{
    OSStatus    osStatus = noErr;
    NavDialogOptions dialogOptions;
    WindowRef    windowRef;
    NavEventUPP    navEventFunctionUPP;
    TXNFileType    txnFileType;
    NavReplyRecord navReplyStruc;
    AEKeyword    theKeyword;
    DescType    actualType;
    FSSpec    fileSpec;
    Size    actualSize;
    AliasHandle    aliasHdl;

    osStatus = NavGetDefaultDialogOptions(&dialogOptions);

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

        GetWTitle(windowRef,dialogOptions.savedFileName);
        GetIndString(dialogOptions.clientName,rMiscellaneousStrings,sApplicationName);

        navEventFunctionUPP = NewNavEventUPP((NavEventProcPtr) navEventFunction);

        GetWindowProperty(windowRef,kFileCreator,'FiTy',sizeof(TXNFileType),NULL,&txnFileType);

        osStatus = NavPutFile(NULL,&navReplyStruc,&dialogOptions,navEventFunctionUPP,
            txnFileType,kFileCreator,NULL);

        DisposeNavEventUPP(navEventFunctionUPP);

        if(navReplyStruc.validRecord && osStatus == noErr)
        {
            if((osStatus = AEGGetNthPtr(&(navReplyStruc.selection),1,typeFSS,&theKeyword,
                &actualType,&fileSpec,sizeof(fileSpec),&actualSize))
                == noErr)
            {
                if(!navReplyStruc.replacing)
                {
                    osStatus = FSpCreate(&fileSpec,kFileCreator,txnFileType,navReplyStruc.keyScript);
                    if(osStatus != noErr)
                    {
                        NavDisposeReply(&navReplyStruc);
                        return osStatus;
                    }
                }
            }

            if(osStatus == noErr)
            {
                SetWTitle(windowRef,fileSpec.name);

                SetWindowProperty(windowRef,kFileCreator,'FiSp',sizeof(FSSpec),&fileSpec);
                SetPortWindowPort(windowRef);
                SetWindowProxyFSSpec(windowRef,&fileSpec);
                GetWindowProxyAlias(windowRef,&aliasHdl);
                SetWindowProperty(windowRef,kFileCreator,'tALH',sizeof(AliasHandle),&aliasHdl);
                SetWindowModified(windowRef,false);

                osStatus = doWriteFile(windowRef,!navReplyStruc.replacing);
            }

            NavCompleteSave(&navReplyStruc,kNavTranslateInPlace);
        }

        NavDisposeReply(&navReplyStruc);
    }
}

if(osStatus == userCanceledErr)

```

```

    osStatus = noErr;

return osStatus;
}

// ***** doRevertCommand

OSStatus doRevertCommand(void)
{
    OSStatus      osStatus = noErr;
    NavDialogOptions  dialogOptions;
    NavEventUPP      navEventFunctionUPP;
    WindowRef        windowRef;
    Str255           fileName;
    NavAskSaveChangesResult reply;
    TXNObject        txnObject = NULL;

    osStatus = NavGetDefaultDialogOptions(&dialogOptions);

    if(osStatus == noErr)
    {
        navEventFunctionUPP = NewNavEventUPP((NavEventProcPtr) navEventFunction);

        windowRef = FrontWindow();
        GetWTitle(windowRef,fileName);
        BlockMoveData(fileName,dialogOptions.savedFileName,fileName[0] + 1);

        osStatus = NavAskDiscardChanges(&dialogOptions,&reply,navEventFunctionUPP,0);

        DisposeNavEventUPP(navEventFunctionUPP);

        if(osStatus == noErr)
        {
            if(reply == kNavAskDiscardChanges)
            {
                if(isApplicationWindow(windowRef,&txnObject))
                {
                    TXNRevert(txnObject);

                    if(TXNDataSize(txnObject)
                       SetWindowModified(windowRef,false);
                }
            }
        }
    }

    return osStatus;
}

// ***** doQuitCommand

OSStatus doQuitCommand(NavAskSaveChangesAction action)
{
    OSStatus osStatus = noErr;

    while(FrontWindow())
    {
        osStatus = doCloseCommand(action);
        if(osStatus != noErr)
            return osStatus;
    }

    return osStatus;
}

// ***** doNewDocWindow

OSStatus doNewDocWindow(WindowRef *outWindowRef,FSSpec *fileSpec,TXNFileType txnFileType)
{
    WindowRef      windowRef;
    Str255          numberAsString, titleString = "\puntitled ";
    Rect            availableBoundsRect, portRect;
    SInt16          windowHeight;
    TXNFrameOptions txnFrameOptions;
    OSStatus        osStatus = noErr;
    TXNObject        txnObject = NULL;
    TXNFrameID       txnFrameID;
    RGBColor        frameColour = { 0xEEEE, 0xEEEE, 0xEEEE };
    TXNControlTag    txnControlTag[1];

```

```

TXNControlData txnControlData[1];
TXNMargins    txnMargins;
CGContextRef  cgContextRef;

//
..... get window

if(!(windowRef = GetNewCWindow(rNewWindow,NULL,(WindowRef) -1)))
    return MemError();
SetPortWindowPort(windowRef);

ChangeWindowAttributes(windowRef,kWindowInWindowMenuAttribute,0);

gUntitledWindowNumber++;
if(gUntitledWindowNumber != 1)
{
    NumToString(gUntitledWindowNumber,numberAsString);
    doConcatPStrings(titleString,numberAsString);
}
SetWTitle(windowRef,titleString);

// ..... extend window bottom to bottom of screen less the
dock

GetAvailableWindowPositioningBounds(GetMainDevice(),&availableBoundsRect);
GetWindowPortBounds(windowRef,&portRect);
LocalToGlobal(&topLeft(portRect));
windowHeight = availableBoundsRect.bottom - portRect.top;
SizeWindow(windowRef,630,windowHeight,false);

// ..... get new TXNObject and
attach window to it

txnFrameOptions = kTXNWantHScrollBarMask | kTXNWantVScrollBarMask | kTXNShowWindowMask;

osStatus = TXNNewObject(fileSpec>windowRef,NULL,txnFrameOptions,kTXNTextEditStyleFrameType,
    txnFileType,kTXNSystemDefaultEncoding,&txnObject,&txnFrameID,
    NULL);

if(osStatus == noErr)
{
    // ..... associate frame ID and TXNObject
with window

    SetWindowProperty(windowRef,kFileCreator,'tOBJ',sizeof(TXNObject),&txnObject);
    SetWindowProperty(windowRef,kFileCreator,'tFRM',sizeof(TXNFrameID),&txnFrameID);
    if(fileSpec != NULL)
        SetWindowProperty(windowRef,kFileCreator,'FiSp',sizeof(FSSpec),fileSpec);
    SetWindowProperty(windowRef,kFileCreator,'FiTy',sizeof(TXNFileType),&txnFileType);

//
..... set margins

    txnControlTag[0] = kTXNMarginsTag;
    txnControlData[0].marginsPtr = &txnMargins;

    txnMargins.leftMargin = txnMargins.topMargin = 10;
    txnMargins.rightMargin = txnMargins.bottomMargin = 10;
    TXNSetTXNObjectControls(txnObject,false,1,txnControlTag,txnControlData);

// ..... create core graphics context and pass
to MLTE

    if(gRunningOnX)
    {
        CreateCGContextForPort(GetWindowPort(windowRef),&cgContextRef);
        txnControlTag[0] = kATSUCGContextTag;
        txnControlData[0].uValue = (UInt32) cgContextRef;
        TXNSetTXNObjectControls(txnObject,false,1,txnControlTag,txnControlData);
    }
}
else
    doErrorAlert(osStatus);

gCurrentNumberOfWindows++;
if(gCurrentNumberOfWindows == 1)
    doEnableDisableMenus(true);

```



```

*outWindowRef = windowRef;

return noErr;
}

// ***** doOpenFile

OSStatus doOpenFile(FSSpec fileSpec, OSType fileType)
{
    OSStatus osStatus = noErr;
    WindowRef windowRef;
    AliasHandle aliasHdl;

    if(osStatus = doNewDocWindow(&windowRef, &fileSpec, fileType))
        return osStatus;

    SetWTitle(windowRef, fileSpec.name);

    SetWindowProxyFSSpec(windowRef, &fileSpec);
    GetWindowProxyAlias(windowRef, &aliasHdl);
    SetWindowProperty(windowRef, kFileCreator, 'tALH', sizeof(AliasHandle), &aliasHdl);

    SetWindowModified(windowRef, false);

    return noErr;
}

// ***** doCloseFile

void doCloseWindow(WindowRef windowRef, TXNObject txnObject)
{
    TXNDeleteObject(txnObject);
    DisposeWindow(windowRef);
    gCurrentNumberOfWindows--;
    if(gCurrentNumberOfWindows == 0)
        doEnableDisableMenus(false);
}

// ***** doWriteFile

OSStatus doWriteFile(WindowRef windowRef, Boolean newFile)
{
    TXNPermanentTextEncodingType encodingType;
    TXNObject txnObject = NULL;
    FSSpec fileSpec, fileSpecTemp;
    TXNFileType txnFileType;
    UInt32 currentTime;
    Str255 tempFileName;
    OSStatus osStatus = noErr;
    SInt16 tempFileVolNum, tempFileRefNum, tempResForkRefNum = -1;
    SInt32 tempFileDirID;
    Boolean hasResFile = false;

    GetWindowProperty(windowRef, kFileCreator, 'tOBJ', sizeof(TXNObject), NULL, &txnObject);
    GetWindowProperty(windowRef, kFileCreator, 'FiSp', sizeof(FSSpec), NULL, &fileSpec);
    GetWindowProperty(windowRef, kFileCreator, 'FiTy', sizeof(TXNFileType), NULL, &txnFileType);

    encodingType = (txnFileType == kTXNTextFile) ? kTXNMacOSEncoding : kTXNUnicodeEncoding;

    GetDateTime(&currentTime);
    NumToString((SInt32) currentTime, tempFileName);

    osStatus = FindFolder(fileSpec.vRefNum, kTemporaryFolderType, kCreateFolder, &tempFileVolNum,
        &tempFileDirID);
    if(osStatus == noErr)
        osStatus = FMakeFSSpec(tempFileVolNum, tempFileDirID, tempFileName, &fileSpecTemp);
    if(osStatus == noErr || osStatus == fnfErr)
        osStatus = FSpCreate(&fileSpecTemp, 'trsh', 'trsh', smSystemScript);
    if(osStatus == noErr)
        osStatus = FSpOpenDF(&fileSpecTemp, fsRdWrPerm, &tempFileRefNum);

    if(osStatus == noErr)
    {
        if(txnFileType == kTXNTextFile)
        {
            FSpCreateResFile(&fileSpecTemp, 'trsh', 'trsh', smSystemScript);
            osStatus = ResError();
            if(osStatus == noErr)

```

```

    tempResForkRefNum = FSpOpenResFile(&fileSpecTemp,fsRdWrPerm);
    hasResFile = true;
}
}

if(osStatus == noErr)
    osStatus = TXNSave(txnObject,txnFileType,kTXNMultipleStylesPerTextDocumentResType,
        encodingType,&fileSpec,tempFileRefNum,tempResForkRefNum);

if(osStatus == noErr)
    osStatus = FSpExchangeFiles(&fileSpecTemp,&fileSpec);
if(osStatus == noErr)
    osStatus = FSpDelete(&fileSpecTemp);

if(osStatus == noErr)
    osStatus = FSClose(tempFileRefNum);
if(osStatus == noErr)
    if(tempResForkRefNum != -1)
        CloseResFile(tempResForkRefNum);

osStatus = ResError();

if(osStatus == noErr)
    if(newFile)
        osStatus = doCopyResources(fileSpec,txnFileType,hasResFile);

return osStatus;
}

// ***** doCopyResources

OSStatus doCopyResources(FSSpec fileSpec,TXNFileType fileType,Boolean hasResFile)
{
    OSStatus osStatus = noErr;
    SInt16 fileRefNum;

    if(!hasResFile)
        FSpCreateResFile(&fileSpec,kFileCreator,fileType,smSystemScript);

    osStatus = ResError();
    if(osStatus == noErr)
        fileRefNum = FSpOpenResFile(&fileSpec,fsRdWrPerm);

    if(fileRefNum > 0)
        osStatus = doCopyAResource('STR ',-16396,gAppResFileRefNum,fileRefNum);
    else
        osStatus = ResError();

    if(osStatus == noErr)
        CloseResFile(fileRefNum);

    osStatus = ResError();

    return osStatus;
}

// ***** doCopyAResource

OSStatus doCopyAResource(ResType resourceType,SInt16 resourceID,SInt16 sourceFileRefNum,
    SInt16 destFileRefNum)
{
    Handle sourceResourceHdl;
    Str255 sourceResourceName;
    ResType ignoredType;
    SInt16 ignoredID;

    UseResFile(sourceFileRefNum);

    sourceResourceHdl = GetResource(resourceType,resourceID);

    if(sourceResourceHdl != NULL)
    {
        GetResInfo(sourceResourceHdl,&ignoredID,&ignoredType,sourceResourceName);
        DetachResource(sourceResourceHdl);
        UseResFile(destFileRefNum);
        AddResource(sourceResourceHdl,resourceType,resourceID,sourceResourceName);
        if(ResError() == noErr)
            UpdateResFile(destFileRefNum);
    }
}

```

```

ReleaseResource(sourceResourceHdl);

return ResError();
}

// ***** navEventFunction

void navEventFunction(NavEventCallbackMessage callBackSelector, NavCBRecPtr callBackParms,
                     NavCallBackUserData callBackUD)
{
WindowRef windowRef;

switch(callBackSelector)
{
case kNavCBEvt:
switch(callBackParms->eventData.eventDataParms.event->what)
{
case updateEvt:
windowRef = (WindowRef) callBackParms->eventData.eventDataParms.event->message;
if(GetWindowKind(windowRef) != kDialogWindowKind)
doUpdate((EventRecord *) callBackParms->eventData.eventDataParms.event);
break;
}
break;
}
}

// *****

```

# **Demonstration Program MLTETextEditor Comments**

---

This program, like all previous programs, demonstrates the programming of one particular aspect of the Mac OS. However, unlike all previous demonstration programs, it can also be used as a useful application, that is, as a fully functional basic text editor.

New documents created by the text editor are created and saved as Textension ('txtn') file types. Existing files of type 'TEXT' and Unicode ('utxt') can be opened and saved as 'TEXT' and Unicode files. For 'TEXT' files, style information is saved in 'styl' resources. Movies may be embedded within documents.

Those areas of the program relating to file operations and window proxy icons follow the same general approach as does the demonstration program Files (Chapter 18). This includes the file synchronisation function and the functions for copying the missing application name string resource to the resource fork of saved files. The Apple event handlers are identical to those in the demonstration program Files, except for the added capability to handle the Print Documents event.

## **MLTETextEditor.c**

---

### **main**

The application's resource fork file reference number is saved for use in the function doCopyResources.

CreateStandardWindowMenu is used to create a Window menu, which is then given an ID and added to the menu list. If the program is running on Mac OS 8/9, the first item in the Window menu (Zoom Window) is deleted. (As will be seen, in this program, TXNZoomWindow is called when the user clicks the zoom box/button. TXNZoomWindow adjusts the scroll bars automatically; however, this does not occur when Zoom Window is chosen from a Window menu. It is thus necessary to delete the item in this particular program.)

After a reference to the Font menu is obtained, TXNNewFontMenuObject is called to create a hierarchical Font menu. Note that the value passed in the third parameter, which specifies the ID of the first sub-menu, must be 160 or higher.

### **doInitializeMLTE**

doInitializeMLTE is called from doPreliminaries. The call to TXNInitTextension initialises the Textension library. Font information specifying that the default font, font size, and font style for the system default encoding (Unicode on systems with ATSUI) be New York, 12 point, normal is passed in the first parameter. The encoding field specifies how the application wants to see text. The third parameter specifies that embedded movies are to be supported.

### **doInstallAEHandlers**

Note that openAndPrintDocsEventHandler will be called when both an Open Documents and a Print Documents event is received, the difference being that the reference constant is set to kOpen (0) for an Open Application event and to kPrint (1) for a Print Documents event.

### **eventLoop**

Note that the value passed in WaitNextEvent's sleep parameter is the value returned by a call to the function doGetSleepTime. Note also that, when a NULL event is returned by WaitNextEvent, the functions doidle and doSynchroniseFiles are called.

### **doGetSleepTime**

doGetSleepTime determines the value passed in WaitNextEvent's sleep parameter.

This is the first of many functions which call the function isApplicationWindow. As will be seen, isApplicationWindow returns true if there is an open window and if it is of the application kind. It also returns, in the txnObject parameter, the TXNObject to which the window was attached when the window was created.

If there is an open window, and if it is of the application kind, TXNGetSleepTicks is called to get the sleep time to be passed to WaitNextEvent. This ensures that the function doidle will be called at the appropriate interval. If the front window is of the dialog kind, the sleep time is set to the value returned by a call to GetCaretTime. (Actually, in this application, which presents no dialogs with edit text items, it might be considered more appropriate to set the sleep time to the maximum unsigned long value at the else statement.)

### **doidle**

doidle is called to perform idle processing.

The call to TXNGetChangeCount gets the number of times the document has been changed since the last time the TXNObject was saved. If any changes have been made since the last save, SetWindowModified is called to disable the window proxy icon.

### **doEvents**

At the keyDown case, note that TXNKeyDown does not need to be called in a Carbon application. Carbon special-cases Command-key events to avoid them being sent to MLTE. However, all other key-downs get sent directly to the Type Services Manager and your application never gets to "see" them. This means that, in Carbon applications, you cannot filter out characters for special handling before they are passed to MLTE (TXNKeyDown) as you could in a Classic application.

The only exception is command-key events; for command keys, because MLTE

has the habit of eating all keystrokes that go to it, even command keys that it can't process, we detect if the command key exists in the menus and special-case it to avoid sending it to MLTE.

At the mouseMovedMessage case within the osEvt case, TXNAdjustCursor is called to handle cursor shape changing. If the mouse is over a text area, TXNAdjustCursor sets the cursor to the I-beam cursor. If the cursor is over a movie, over a scroll bar, or outside a text area, TXNAdjustCursor sets the cursor to the arrow cursor.

### ***doMouseDown***

At the inGrow case, TXNGrowWindow is called to handle the resizing operation. At the inZoomIn/inZoomOut case, TXNZoomWindow is called to zoom the window. At the inDrag, inGrow, and inZoomIn/inZoomOut cases, TXNAdjustCursor is called after the window has been dragged, re-sized, or zoomed so that the mouse-moved region is re-calculated.

### ***doActivate***

As will be seen, when TXNObject is created and a window attached to it, SetWindowProperty is called to associate the TXNObject frame ID with the window. The call to GetWindowProperty retrieves this frame ID.

If the window is becoming active, TXNActivate is called, with true passed in the third parameter, to activate the scroll bars. Also, TXNFocus is called, with true passed in the second parameter to activate text input (selection and typing). If the window is becoming inactive, false is passed in TXNActivate's third parameter and TXNFocus' second parameter to deactivate the scroll bars and text input.

### ***doUpdate***

TXNUpdate is called to redraw everything in the content area. Note that this function calls BeginUpdate and EndUpdate, so there is no necessity for the application to do so.

### ***isApplicationWindow***

isApplicationWindow is called from many functions. It returns true if there is a front window and if that window is of the application kind. It also returns to the caller the TXNObject to which that window is attached. As will be seen, the TXNObject is associated with the window when both are created, and is retrieved here by the call to GetWindowProperty.

### ***doSynchroniseFiles***

doSynchroniseFiles is the file synchronisation function (see Chapter 18). It is adapted from the function of the same name in the demonstration program Files. In this version:

- The method used to determine whether the window has a file associated with it is to call GetWindowProperty in an attempt to retrieve the handle to the alias structure which, as will be seen, is associated with a window by a call to SetWindowProperty when a file is saved or loaded.
- If the aliasChanged parameter is set to true in the call to ResolveAlias, meaning that the location of the file has changed, the file system specification structure returned by ResolveAlias is associated with the window by the call to SetWindowProperty, replacing the previous file system specification structure stored in the window.
- At the inner if statement, if the file is found to be in the trash, GetWindowProperty is called to return the TXNObject associated with the window when it was created, TXNDeleteObject is called to delete the TXNObject and its associated data structures, and DisposeWindow is called to dispose of the window.

### ***openAndPrintDocsEventHandler***

openAndPrintDocsEventHandler is called when an Open Documents or Print Documents Apple event is received. In both cases, doOpenFile is called to open and display the file. In the case of a Print Documents event, TXNPrint is also called to print the document, following which doCloseCommand is called to dispose of the window and its TXNObject.

### ***doErrorAlert***

If the error code is kATSUFonTSMatched (-8793), doErrorAlert simply returns. kATSUFonTSMatched is not an error as such. It but is returned by ATSUMatchFontsToText when changes need to be made to the fonts associated with the text.

## ***MLTEMenus.c***

---

### ***doEnableDisableMenus***

doEnableDisableMenus is called from doCloseWindow and doNewDocWindow to ensure that all menus except the File menu are disabled if no windows are open and that those menus are enabled if at least one window is open.

### ***doAdjustFileMenu***

If the call to TXNGetChangeCount reveals that the document has been changed since it was opened or last saved, the File menu Save and Revert items are enabled, otherwise they are disabled.

If the call to TXNDataSize reveals that there are characters in the TXNObject, the Save As, Page Setup, and Print items are enabled, otherwise they are disabled.

The else block executes only if no windows are open, ensuring that all File menu items except New, Open, and Quit are disabled.

## **doEditMenu**

At the first block, all Edit menu items are disabled. At the second block, the default item text for both the Undo and Redo items (Can't Undo, Can't Redo) is set. This may be changed by the next two blocks.

The next block addresses the Undo item. The call to TXNCanUndo determines whether the last action is undoable. If the last action is undoable, TXNCanUndo returns, in the second parameter, an action key code which will be used to index a STR# resource for a string describing the undoable action. If this action key code represents a typing, cut, paste, clear, change font, change font colour, change font size, change font style, change alignment, drag action, or move action, the appropriate string is retrieved by the call to GetIndString and the item text is set to this string (for example, "Undo Cut"). If the action is any other action, the item text is set to "Undo".

At the block beginning with the call to TXNCanRedo, the same process is repeated in respect of the Redo item.

If the call to TXNIsSelectionEmpty reveals that the current selection is not empty, the Cut, Copy, and Clear items are enabled.

If the call to TXNIsScrapPastable reveals that the current scrap contains data that is supported by MLTE, the Paste item is enabled.

If the call to TXNDataSize reveals that there are characters in the TXNObject, the Select All item is enabled.

## **doPrepareFontMenu**

doAdjustFileMenu and doAdjustEditMenu are concerned with enabling and disabling menu items as appropriate. doPrepareFontMenu and the other menu preparation functions are concerned with adding and removing checkmarks from items.

For the Font menu, all that is required is a call to TXNPrepareFontMenu. If the insertion point caret is in text in a particular font, or if a selection contains text in a single font, that menu item will be checkmarked. (If the font is in a Font menu sub-menu, the item in the sub-menu is checkmarked and a "dash" marking character is placed in the Font menu item to which the sub-menu is attached.) On the other hand, if a selection contains text in more than one font, all marking characters are removed from the Font menu and its sub-menus.

## **doPrepareSizeMenu**

doPrepareSizeMenu does for the Size menu what doPrepareFontMenu does for the Font menu. If the insertion point caret is in text in a particular size, or if a selection contains text in a single size, the associated Size menu item is checkmarked. On the other hand, if a selection contains text in more than one size, all Size menu items are un-checkmarked.

Font size is represented by a value of type Fixed (four bytes comprising 16-bit signed integer plus 16-bit fraction). Accordingly, the itemSizes array is initialised with the sizes represented in the Size menu (9, 10, 11, 12, 14, 18, 24, 36) expressed as Fixed values. Each element in the array corresponds to an individual menu item.

The tag and size fields of a structure of type TXNTypeAttributes are assigned, respectively, a value ('size') specifying the size attribute and the size of the Fixed data type. The call to TXNGetContinuousTypeAttributes tests the current selection to see if the font size is continuous. On output, bit 1 in the second parameter (txContinuousFlags) will be set if all the text in the selection is all of one size, and the dataValue field of the data field of txnTypeAttributes will contain that size.

All items in the menu are then walked. If bit 1 of txContinuousFlags is not set, all items will be un-checkmarked. If bit 1 is set, and if the font size returned in the dataValue field is equal to the value in that element of the itemSizes array corresponding to the current menu item, that item is checkmarked, otherwise it is un-checkmarked.

## **doPrepareStyleMenu**

The same general approach is used to prepare the Style menu. In this case, the tag and size fields of a structure of type TXNTypeAttributes are assigned, respectively, a value ('face') specifying the style attribute and the size of the Style data type. Also, the block which checkmarks or un-checkmarks the menu items is a little different, reflecting the fact that the bold, italic, and underline styles can be cumulative.

The first call to CheckMenuItem checkmarks the Plain menu item if all the text in the selection is of the same style and if that style is the plain (normal) style. The for loop addresses the bold, italic, and underline menu items only. If all the text in the selection is of the same style, or combination of styles, the menu item/s corresponding to the bits set in the dataValue field of the data field of txnTypeAttributes is/are checkmarked, otherwise, it/they is/are uncheckmarked.

## **doPrepareColourMenu**

doPrepareColourMenu is similar to doAdjustSizeMenu except that the tag and size fields of a structure of type TXNTypeAttributes are assigned, respectively, a value ('klor') specifying the colour attribute and the size of the RGBColor data type. Note also that the address of the attributesColour variable is assigned to the dataPtr field of the data field of txnTypeAttributes, meaning that attributesColour receives the colour returned by the call to TXNGetContinuousTypeAttributes. It is the colour stored in attributesColour that is compared with the colours stored in the itemColours array in order to determine whether a menuItem should be checked or unchecked (assuming that the selection contains text in one colour only).

## **doPrepareJustificationMenu**

In doPrepareJustificationMenu, the first element of a single-element array of type TXNControl is assigned the control tag 'just'. The call to TXNGetTXNObjectControls returns, in the uValue field of the first element of a single-element array of type TXNControlData, a value representing the current justification setting in the TXNObject. This value determines which item in the Justification menu is checkmarked, all other items being uncheckmarked.

## ***doFileMenuChoice***

doFileMenuChoice is broadly similar to the function of the same name in the demonstration program Files, except as follows.

At the iPageSetup case, TXNPageSetup is called. TXNPageSetup displays the Page Setup dialog and handles all text re-formatting arising from user interaction with the dialog.

At the iPrint case, TXNPrint is called. TXNPrint displays the Print dialog and prints the document.

At the iQuit case, if close-down has not been interrupted by the user clicking in the Cancel button of a Save Changes dialog, TXNDisposeFontMenuObject is called to dispose of the TXNFontMenuObject created at program start. Note that, even if the object is successfully disposed of, it is still necessary to set the associated global variable to NULL.

## ***doEditMenuChoice***

At the iUndo and iRedo cases, TXNUndo and TXNRedo are called to undo and redo the last action.

At the iCut and iCopy cases, TXNCut and TXNCopy are called to cut and copy the current selection to MLTE's private scrap. TXNConvertToPublicScrap is also called to copy MLTE's private scrap to the public scrap (clipboard). Note that, for reasons explained at Chapter 20, TXNConvertToPublicScrap must not be called at a suspend event in a Carbon application.

At the iPaste case, TXNPaste is called to paste MLTE's private scrap to the document. Note that there is no need to precede this call with a call to TXNConvertFromPublicScrap in a Carbon application. In a Carbon application, MLTE keeps the public scrap (clipboard) synchronised with MLTE's private scrap.

At the iClear case, TXNClear is called to delete the current selection without copying it to the MLTE private scrap. At the iSelectAll case, TXNSelectAll is called to select everything in the frame.

## ***doFontMenuChoice***

doFontMenuChoice handles choices from the Font menu. The call to TXNFontMenuSelection takes a menu ID and menu item index and changes the current selection to the font represented by that menu item.

## ***doSizeMenuChoice***

doSizeMenuChoice handles choices from the Size menu. The received menu item index is used to determine which element of the itemSizes array is assigned to the variable sizeToSet. The tag and size fields of a structure of type TXNTypeAttributes are then assigned, respectively, a value ('size') specifying the size attribute and the size of the Fixed data type. The dataValue field of the data field is assigned the size to set. The call to TXNSetTypeAttributes sets the font size in the specified TXNObject.

## ***doStyleMenuChoice and doColourMenuChoice***

doStyleMenuChoice and doColourMenuChoice handle choices from the Style and Colour menus, and use the same general approach as doSizeMenuChoice. The exception is that, in doColourMenuChoice, if the Colour Picker item is chosen, GetColor is called to present the Color Picker dialog to solicit a colour choice by the user.

## ***doJustificationMenuChoice***

doJustificationMenuChoice handles choices from the Justification menu. The received menu item index is used to determine which element of the itemJustification array is assigned to the variable justificationToSet. The single element of an array of type TXNControlTag is then assigned a value ('just') specifying the justification tag. The call to TXNGetTXNObjectControls returns, in the fourth parameter, the TXNObject's current justification setting. If this setting is not the same as the justification the user is attempting to set, TXNSetTXNObjectControls is called to set the chosen justification in the TXNObject. false is passed in the second parameter so that all controls are not reset to the defaults.

## ***MLTENewOpenCloseSave.c***

---

The file handling functions in this section are broadly similar to those in the demonstration program Files. This includes those areas of the code relating to window proxy icons. Accordingly, generally speaking, only the code which differs from the Files code is explained in the following.

Fig 1 shows the general File menu and Apple event handling strategy, as adapted from Fig 4 at Chapter 18.

## ***doNewCommand***

doNewCommand is called when the user chooses New from the File menu, and from the Open Application and Re-Open Application Apple Event handlers.

If the call to doNewDocWindow, a reference to the created window will be returned in the first parameter, NULL is passed in the second (file system specification) parameter, and the third parameter specifies the required file type as Textension. (Note: If you prefer the file type for documents created by the program to be TEXT or Unicode, the only actions required are to pass kTXNTextFile or kTXNUnicodeTextFile in the third parameter of the calls to doNewDocWindow and SetWindowProxyCreatorAndType.)

## ***doOpenCommand***

doOpenCommand is called when the user chooses Open from the File menu. Recall that the aim is to get the file system specification and file type for the file, or files, selected in the Navigation Services Open dialog and pass them in a call to doOpenFile.

## doCloseCommand

doCloseCommand is called when the user chooses Close from the File menu, when the user clicks the go-away box of a window, and for each open window when the user chooses Quit from the File menu or the Quit Application Apple event handler is invoked.

if the call to TXNGetChangeCount reveals that no changes have been made to the document since it was opened, or since the last save, doCloseWindow is called. If changes have been made, a Navigation Services Save Changes dialog box is presented. If the user clicks the Save button, doSaveCommand and then doCloseWindow are called. If the user clicks the Don't Save button, doCloseWindow is called. If the user clicks the Cancel button, that fact is simply reported to the calling function and no other action is taken.

## doSaveCommand

doSaveCommand is called when the user chooses Save from the File menu and by doCloseCommand if the user clicks the Save button in the Save Changes dialog box.

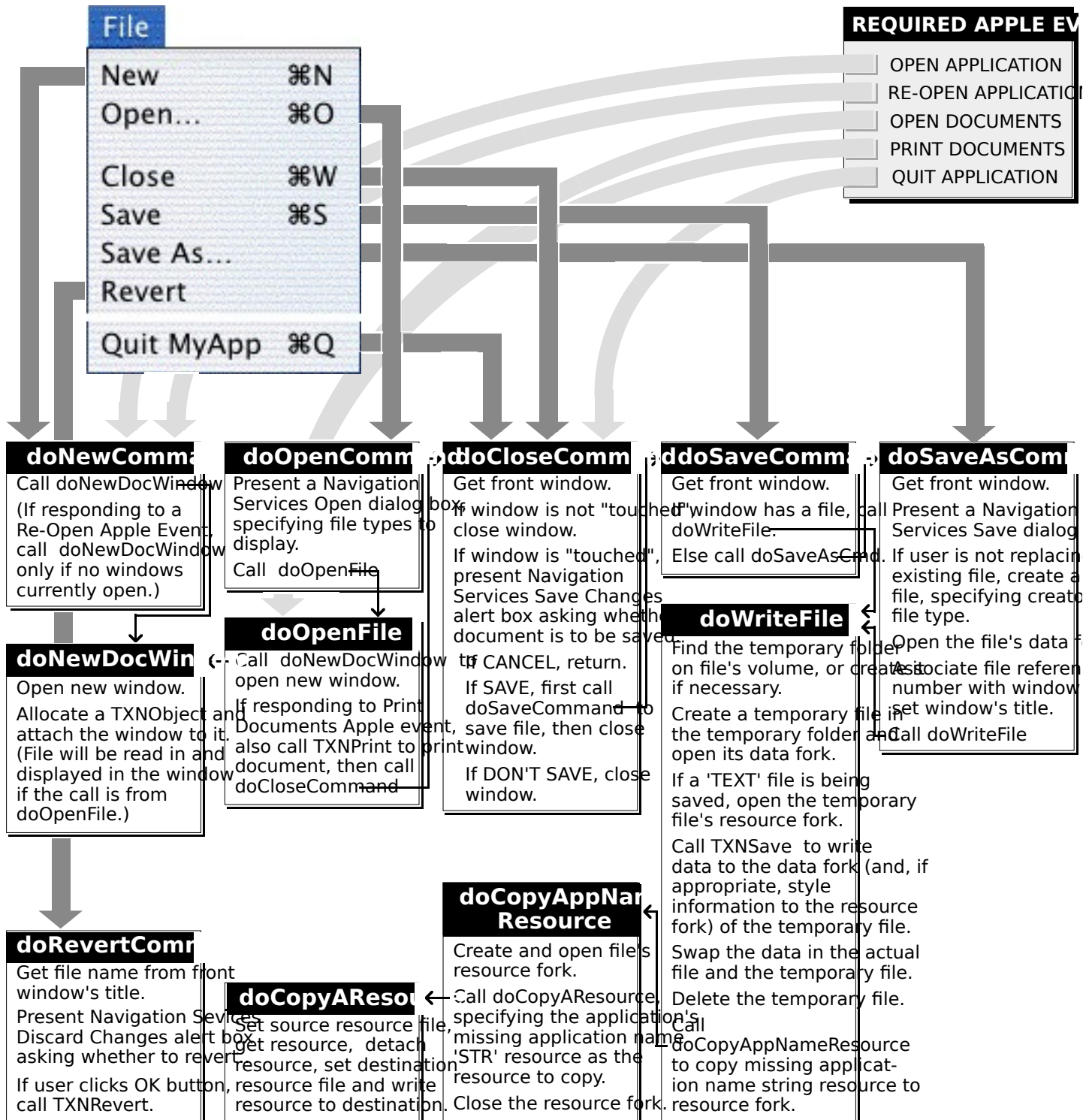


FIG 1 - GENERAL FILE MENU AND REQUIRED APPLE EVENTS HANDLING STRATEGY - N

As will be seen, a file system specification is only associated with a window when an existing document is opened or a new document is saved. Thus the call to GetWindowProperty will return false if a new document has not yet been saved. In this case, doSaveAsCommand is called to solicit a filename from the user and then save the document to that file by calling



doWriteFile. If a file system specification is already associated with the window, the call to doSaveAsCommand is bypassed and doWriteFile is called to save the file to the existing filename.

### **doSaveAsCommand**

doSaveAsCommand is called when the user chooses SaveAs from the File menu and from doSaveCommand when the front window does not yet have a file associated with it.

As will be seen, the file type is associated with the window when the window is created. The call to GetWindowProperty retrieves the file type so that it can be passed in the fifth parameter of the call to NavPutFile and, if the file is not being replaced, in the third parameter of the call to FSpCreate.

The two calls to SetWindowProperty associates the file system specification returned by AEGGetNthPtr, and a handle to the alias data for the file returned by the call to GetWindowProxyAlias, with the window. (The latter is used by the file synchronisation function.)

The call to doWriteFile writes the file.

### **doRevertCommand**

If, when the Discard Changes dialog box is presented, the user clicks the OK button, TXNRevert is called to revert to the last saved version of the document or, if the file was not previously saved, to revert to an empty document. The call to TXNDataSize determines whether the revert has been to an empty document. If not, SetWindowModified is called with false passed in the modified parameter to cause the window proxy icon to appear in the enabled state, indicating no unsaved changes.

### **doQuitCommand**

doQuitCommand is called when the user chooses Quit from the File menu. For each open window, doCloseCommand is called.

### **doNewDocWindow**

doNewDocWindow creates a new window, creates a new TXNObject and attaches the window to it, associates certain information with the window, sets the background to a light grey colour (for demonstration purposes), and sets the margins.

GetNewCWindow creates a new invisible window and SetPortWindowPort makes its graphics port the current port. The call to ChangeWindowAttributes ensures that the window's title will appear as an item in the Window menu.

The next block adjusts the height of the (invisible) window so that the bottom is just above the space occupied by the Mac OS 8/9 control strip.

Preparatory to the call to TXNNewObject, a variable of type TXNFrameOptions is assigned a value which will specify that the created TXNObject is to support horizontal and vertical scroll bars and that the window should be displayed before the call to TXNNewObject returns.

The call to TXNNewObject creates a new TXNObject and attaches it to the window specified in the second parameter. NULL is passed in the third (iFrame) parameter, meaning that the window's port rectangle will be used as the frame. Note that kTXNTextensionFile will be passed in the sixth (iFileType) parameter if a new document is being created, and that kTXNTextFile or kTXNUnicodeTextFile will be passed in if a 'TEXT' or Unicode file is being opened. Note also that the local variable txnFrameID will contain the frame ID when TXNNewObject returns.

If a pointer to file system specification structure is passed in TXNNewObject's first parameter, TXNNewObject will read in the file and display its contents. If NULL is passed in this parameter, the document will start empty. (Recall that NULL will be received in the fileSpec formal parameter when doNewDocWindow is called from doNewCommand, and a pointer to a file system specification structure will be received when doNewDocWindow is called from doOpenFile.)

The next block associates the TXNObject, the frame ID, and the received file system specification (if any) and file type with the window.

The next block sets the margins to ten pixels all round. The first element of a single-element array of type TXNControlTag is assigned a value ('marg') specifying the margins tag, and the marginsPtr field in the first element a single-element array of type TXNControlData is assigned the address of a local variable of type TXNMargins. The four fields of the TXNMargins structure are then assigned the value 10. The call to TXNSetTXNObjectControls sets the margins.

By default, MLTE renders text via QuickDraw on Mac OS X. The appearance of text on Mac OS X is greatly enhanced by rendering via Core Graphics. Accordingly, a Core Graphics context is created and the information is passed to MLTE by calling TXNSetTXNObjectControls with the kATSUCGContextTag.

If the window and TXNObject were successfully created, the global variable which keeps track of the number of open windows is incremented. If the previous number of open windows was zero, meaning that all of the applications menus less the Apple/Application and File menus would have been disabled, doEnableDisableMenus is called to enable those menus.

### **doOpenFile**

doOpenFile is called from doOpenCommand and from the Open Documents Apple event handler. The received file specification structure and file type are passed in a call to doNewDocWindow to open a window and create a TXNObject. Since the file system specification structure will be passed in the call to TXNNewObject in doNewDocWindow, TXNNewObject will read in the file and display its contents.

## ***doCloseWindow***

When `doCloseWindow` is called to close a window, `TXNDeleteObject` is called to delete the `TXNObject` and all associated data structures. The window is then disposed of, and the global variable which keeps track of the number of open windows is decremented. If no windows remain open, `doEnableDisableMenus` is called to disable all of the applications windows less the Apple/Application and File menus.

## ***doWriteFile***

`doWriteFile` is called by `doSaveCommand` and `doSaveAsCommand`. As in the demonstration program `Files`, a "safe save" procedure is used to save files.

The three calls to `GetWindowProperty` retrieve the `TXNObject`, file system specification, and file type stored in the window object. At the next line, the encoding is set to Mac OS Encoding if the file type is 'TEXT', otherwise encoding is set to Unicode.

After the temporary file is created and its data fork is opened, and if the file type is of type 'TEXT', the resource fork is also created and opened.

The call to `TXNSave` then writes the contents of the document to the temporary file. Note that the file type is passed in the second parameter. If the file is of type 'TEXT', `kTXNMultipleStylesPerTextDocumentResType` passed in the third parameter ensures that style information will be saved to the resource fork as a 'styl' resource.

The call to `FSpExchangeFiles`, swaps the files' data by changing the information in the volume's catalog. The temporary file is then deleted, following which the file's data and resource forks are closed.

If this is a new file, `doCopyResources` is called to copy the missing application name string resource from the resource fork of the application file to the resource fork of the new file.