Multilingual Text Engine Reference

Carbon > Text & Fonts

2007-02-19

Contents

Tables

Multilingual Text Engine Reference

Framework: Carbon/Carbon.h

Declared in HITextViews.h
MacTextEditor.h

Overview

Multilingual Text Engine (MLTE) is an API that allows your application to provide Carbon-compliant Unicode text editing. This document is relevant for anyone who is writing an application that needs to display static Unicode text or provide Unicode-compliant text editing fields. You can also use MLTE if your application provides text editing support within a full-size window. For more information about basic text processing and using MLTE, see the document "Handling Unicode Text Editing with MLTE."

Carbon fully supports the Multilingual Text Engine, and Apple recommends that you adopt Multilingual Text Engine as a replacement for TextEdit.

Functions by Task

Displaying Static Text

`TXNDrawUnicodeTextBox` (page 42)
 Draws a Unicode string in the specified rectangle.

`TXNDrawCFStringTextBox` (page 41)
 Draws text from a Core Foundation string (`CFString`) in the specified rectangle.

Initializing and Terminating MLTE

`TXNInitTextension` (page 62)
 Initializes MLTE.

`TXNVersionInformation` (page 85)
 Gets the version number of MLTE and the set of features in this version.

Working With MLTE Objects

`TXNCreateObject` (page 35)
: Creates a new MLTE text object which is an opaque structure that handles text formatting at the document level.

`TXNGetAccessibilityHIObject` (page 48)
: Obtains an `HIObjectRef` representing the MLTE object for accessibility purposes.

`TXNDeleteObject` (page 38)
: Deletes a previously allocated text object.

`TXNDataSize` (page 37)
: Reports the amount of memory used to hold the text in a given text object.

`TXNAttachObjectToWindowRef` (page 29)
: Attaches a text object to a window.

`TXNGetWindowRef` (page 60)
: Returns a reference to the window to which the specified text object is attached.

Responding to Events

`TXNGetEventTarget` (page 54)
: Obtains the current event target for a TXNObject.

`TXNSetEventTarget` (page 75)
: Sets a Carbon Event target for MLTE Carbon Event handlers.

`TXNGetCommandEventSupport` (page 50)
: Obtains the command event support that is currently set for an MLTE object.

`TXNSetCommandEventSupport` (page 73)
: Enables and disables support for menu commands in MLTE.

`TXNAdjustCursor` (page 28)
: Obtains the current cursor position and draws the cursor in a form appropriate to the content over which it is placed.

`TXNClick` (page 33)
: Processes a mouse-down event in a window’s content area.

`TXNEchoMode` (page 43)
: Determines whether a specified character is drawn instead of the glyph associated with the input character.

`TXNFocus` (page 47)
: Changes the focus of a text object.

`TXNForceUpdate` (page 47)
: Forces an update of the view rectangle and the scroll bars.

`TXNGetSleepTicks` (page 58)
: Reports the appropriate amount of time to allot to background processes, depending on the state of the window.

`TXNIdle` (page 62)
: Does idle time processing, such as flashing the cursor.

`TXNKeyDown` (page 64)
: Processes a key-down event.

`TXNUpdate` (page 85)
: Redraws everything in a frame in response to an update event.

`TXNScroll` (page 70)
: Scrolls the text within a view rectangle of the specified text object.

`TXNRegisterScrollInfoProc` (page 68)
: Installs or uninstalls a scrolling callback function on a text object.

Working With HITextView

`HITextViewCreate` (page 22)
: Creates an HITextView that is initially invisible.

`HITextViewGetTXNObject` (page 23)
: Obtains the text object associated with an HITextView.

`HITextViewCopyBackgroundColor` (page 21)
: Obtains the background color of the view.

`HITextViewSetBackgroundColor` (page 24)
: Sets the background color of the view.

Editing Data

`TXNDrawObject` (page 42)
: Draws a text object in the last window set by your application.

`TXNClear` (page 32)
: Deletes the current selection.

`TXNCopy` (page 33)
: Copies the current selection to the private MLTE scrap.

`TXNCut` (page 37)
: Deletes the current selection and copies it to the private MLTE scrap.

`TXNIsScrapPastable` (page 63)
: Tests whether the Clipboard contains data that is supported by MLTE.

`TXNPaste` (page 66)
: Pastes the contents of the private MLTE scrap into the text object.

`TXNGetData` (page 52)
: Copies a range of data.

`TXNGetDataEncoded` (page 53)
: Copies the text in a specified range, and if necessary, translates the text to match your application's preferred encoding.

`TXNSetData` (page 74)
: Replaces a range of data (text, graphics, and so forth).

`TXNCanRedoAction` (page 30)
: Indicates whether an action can be redone.

`TXNRedo` (page 68)
: Redoes the last command.

`TXNCanUndoAction` (page 31)
: Indicates whether an action can be undone.

`TXNUndo` (page 84)
: Undoes the last command.

Managing Fonts and Font Menus

`TXNCountRunsInRange` (page 34)
: Obtains a count of the style runs in a range of data.

`TXNGetIndexedRunInfoFromRange` (page 55)
: Gets information about a run in a range of data.

`TXNSetTypeAttributes` (page 82)
: Sets text attributes (such as size and style) for the current selection or the text defined by a range you specify.

`TXNGetContinuousTypeAttributes` (page 50)
: Checks to see if the attributes of the current selection are continuous.

`TXNDisposeFontMenuObject` (page 183) Deprecated in Mac OS X v10.5
: Disposes of a Font menu object.

`TXNDoFontMenuSelection` (page 184) Deprecated in Mac OS X v10.5
: Changes the font of the current selection.

`TXNGetFontMenuHandle` (page 185) Deprecated in Mac OS X v10.5
: Gets the Font menu handle that belongs to a Font menu object.

`TXNNewFontMenuObject` (page 185) Deprecated in Mac OS X v10.5
: Creates a new Font menu object.

`TXNPrepareFontMenu` (page 186) Deprecated in Mac OS X v10.5
: Prepares a Font menu for display.

`TXNGetFontDefaults` (page 179) Deprecated in Mac OS X v10.4
: Makes a copy of the font descriptions for a given text object.

`TXNSetFontDefaults` (page 183) Deprecated in Mac OS X v10.4
: Specifies the font descriptions for each script used in a text object.

Managing Layout and Formatting

`TXNRecalcTextLayout` (page 68)
: Recalculates the text layout based on new view and destination rectangles.

`TXNSetTXNObjectControls` (page 81)
: Sets formatting and privileges attributes (such as justification, line direction, tab values, and read-only status) that apply to the entire text object.

`TXNGetTXNObjectControls` (page 59)
: Gets the current formatting and privileges attributes (such as justification, line direction, tab values, and read-only status) for a text object.

`TXNGetLineCount` (page 56)
: Gets the total number of lines in a text object.

`TXNGetLineMetrics` (page 57)
: Gets information about line width and height for a specified line of data in a text object.

`TXNHIPointToOffset` (page 61)
: Gets the offset value that corresponds to a point in local coordinates.

`TXNOffsetToHIPoint` (page 65)
: Obtains the local coordinates of the point that corresponds to a specified offset of a text object.

Managing Selections

`TXNGetSelection` (page 57)
: Gets the absolute offsets of the current selection.

`TXNIsSelectionEmpty` (page 63)
: Determines whether the current selection is empty.

`TXNSelectAll` (page 71)
: Selects all data in the frame of a text object.

`TXNSetSelection` (page 80)
: Specifies the selection range or the position of the insertion point.

`TXNShowSelection` (page 83)
: Scrolls the current selection into view.

Controlling the Frame and Window

`TXNGetViewRect` (page 60)
: Gets the rectangle that describes the current view of the document.

`TXNGrowWindow` (page 61)
: Adjusts the size of a window in response to mouse-down events in the size region of the window.

`TXNResizeFrame` (page 69)
: Resizes the view and destination rectangles.

`TXNSetBackground` (page 72)
: Sets the background on which the text object's data is drawn.

`TXNSetFrameBounds` (page 77)
: Changes the boundaries of a text object's frame.

`TXNZoomWindow` (page 87)
: Increases the size of the data displayed in a window in response to a click in the zoom box.

`TXNSetHIRectBounds` (page 78)
: Sets the view rectangle and/or the destination rectangle.

`TXNGetHIRect` (page 54)
: Obtains the values for the current view, destination, or text rectangle.

`TXNSetScrollbarState` (page 79)
: Sets the state of the scroll bars so they are drawn correctly in response to activate events.

Searching

`TXNFind` (page 45)
: Finds a piece of text or a graphics, sound, or movie object.

Managing Files

`TXNFlattenObjectToCFDataRef` (page 46)
: Flattens a text object so it can be saved to disk or embedded with other data.

`TXNRevert` (page 70)
: Reverts to the last saved version of a document.

`TXNReadFromCFURL` (page 67)
: Reads data from a `CFURLRef` into a TXNObject.

`TXNCopyTypeIdentifiersForRange` (page 34)
: Obtains an array of universal type identifiers for a TXNObject.

`TXNWriteRangeToCFURL` (page 86)
: Writes a range of a text object to a file or to a special file bundle.

Printing

`TXNPageSetup` (page 65)
: Displays the Page Setup dialog for the current default printer and manages changes, such as reformatting the text, in response to page layout changes.

`TXNPrint` (page 66)
: Prints the document so it is formatted to fit the page size selected for the printer.

Supporting Drag and Drop

`TXNDragReceiver` (page 38)
: Handles dragged data in a text object for which a custom drag handler is already in place.

`TXNDragTracker` (page 39)
: Handles tracking a drag event in a text object for which a custom drag handler is already in place.

Keeping Track of User Actions

`TXNGetChangeCount` (page 49)
: Retrieves the number of times a document has been changed.

`TXNGetCountForActionType` (page 51)
: Gets the number of times a given type of action has occurred.

`TXNClearCountForActionType` (page 32)
: Sets the counter for the specified action type to zero.

`TXNBeginActionGroup` (page 29)
: Starts an action group.

TXNEndActionGroup (page 44)
    Ends an action group.

TXNSetActionNameMapper (page 72)
    Sets a callback that MLTE uses to obtain the localized string representing an action or an action group.

Managing Spell Checking As You Type

TXNGetSpellCheckAsYouType (page 58)
    Determines whether the “Spell Check as You Type” feature is enabled.

TXNSetSpellCheckAsYouType (page 81)
    Enables and disables the “Spell Check as You Type” feature.

Working with the Contextual Menu

TXNSetContextualMenuSetup (page 74)
    Provides a callback function that is called before MLTE displays its contextual menu.

Working With UPP Pointers for MLTE Callback Functions

NewTXNActionNameMapperUPP (page 26)
    Creates a new universal procedure pointer (UPP) to an action name mapper callback function.

InvokeTXNActionNameMapperUPP (page 24)
    Calls your action name mapper callback function.

DisposeTXNActionNameMapperUPP (page 20)
    Disposes of the universal procedure pointer (UPP) to your action name mapper callback function.

NewTXNContextualMenuSetupUPP (page 27)
    Creates a new universal procedure pointer (UPP) to a contextual menu setup callback function.

InvokeTXNContextualMenuSetupUPP (page 25)
    Calls your contextual menu setup callback function.

DisposeTXNContextualMenuSetupUPP (page 20)
    Disposes of the universal procedure pointer (UPP) to your contextual menu setup callback function.

NewTXNFindUPP (page 27)
    Creates a new universal procedure pointer (UPP) to a find callback function that uses your criteria for matching.

InvokeTXNFindUPP (page 25)
    Calls your find callback function.

DisposeTXNFindUPP (page 20)
    Disposes of the universal procedure pointer (UPP) to your find callback function.

NewTXNScrollInfoUPP (page 28)
    Creates a new universal procedure pointer (UPP) to a scrolling callback function.

InvokeTXNScrollInfoUPP (page 26)
    Calls your scrolling callback function.

`DisposeTXNScrollInfoUPP` (page 21)
Disposes of the universal procedure pointer (UPP) to your scrolling callback function.

Not Recommended

This section lists functions that are not recommended and you should no longer use. The Carbon Porting Notes for each function provide information on what you should do in place of using the function.

`TXNActivate` (page 164) Deprecated in Mac OS X v10.3
Sets the state of the scroll bars so they are drawn correctly in response to activate events. (Deprecated. Use `TXNSetScrollbarState` (page 79) instead.)

`TXNAttachObjectToWindow` (page 165) Deprecated in Mac OS X v10.3
Attaches a text object to a window. (Deprecated. Use `TXNAttachObjectToWindowRef` (page 29) instead.)

`TXNConvertFromPublicScrap` (page 166) Deprecated in Mac OS X v10.3
Converts the Clipboard content to the private MLTE scrap. (Deprecated. This function isn't needed in Mac OS X.)

`TXNConvertToPublicScrap` (page 166) Deprecated in Mac OS X v10.3
Converts the private MLTE scrap content to the Clipboard. (Deprecated. This function isn't needed in Mac OS X.)

`TXNDraw` (page 167) Deprecated in Mac OS X v10.3
Redraws the text area, including any scroll bars associated with the text frame. (Deprecated. Use the `TXNDrawObject` (page 42).)

`TXNGetRectBounds` (page 167) Deprecated in Mac OS X v10.3
Obtains the values for the current view, destination, and text rectangles. (Deprecated. Use `TXNGetHIRect` (page 54) instead.)

`TXNIsObjectAttachedToSpecificWindow` (page 168) Deprecated in Mac OS X v10.3
Determines whether a text object is attached to a specified window. (Deprecated. Use `TXNGetWindowRef` (page 60) instead.)

`TXNIsObjectAttachedToWindow` (page 169) Deprecated in Mac OS X v10.3
Checks to see if a text object is attached to a window. (Deprecated. Use `TXNGetWindowRef` (page 60) instead.)

`TXNNewObject` (page 170) Deprecated in Mac OS X v10.3
Creates a new MLTE text object which is an opaque structure that handles text formatting at the document level. (Deprecated. Use `TXNCreateObject` (page 35) instead.)

`TXNOffsetToPoint` (page 172) Deprecated in Mac OS X v10.3
Gets the local coordinates of the point that corresponds to a specified offset of a text object. (Deprecated. Use `TXNOffsetToHIPoint` (page 65) instead.)

`TXNPointToOffset` (page 172) Deprecated in Mac OS X v10.3
Gets the offset value that corresponds to a point in local coordinates. (Deprecated. Use `TXNHIPointToOffset` (page 61) instead.)

`TXNSetDataFromFile` (page 173) Deprecated in Mac OS X v10.3
Replaces a range of data with the contents of a file. (Deprecated. Use `TXNSetDataFromCFURLRef` (page 182) instead.)

`TXNSetRectBounds` (page 174) Deprecated in Mac OS X v10.3
Set the view rectangle and/or the destination rectangle. (Deprecated. Use `TXNSetHIRectBounds` (page 78) instead.)

`TXNTerminateTextension` (page 175) Deprecated in Mac OS X v10.3
: Closes the MLTE library. (Deprecated. This function is no longer needed.)

`TXNSetViewRect` (page 163) Deprecated in Mac OS X v10.2
: Sets the rectangle that describes the current view into the document; changes the amount of text that is viewable. (Deprecated. Use `TXNSetFrameBounds` (page 77) or `TXNSetRectBounds` (page 174) instead.)

`DisposeTXNActionKeyMapperUPP` (page 175) Deprecated in Mac OS X v10.4
: Disposes of the universal procedure pointer (UPP) to your action key mapping callback function. (Deprecated. Use `TXNActionNameMapperProcPtr` instead.)

`InvokeTXNActionKeyMapperUPP` (page 176) Deprecated in Mac OS X v10.4
: Calls your action key mapping callback function. (Deprecated. Use `TXNActionNameMapperProcPtr` instead.)

`NewTXNActionKeyMapperUPP` (page 176) Deprecated in Mac OS X v10.4
: Creates a new universal procedure pointer (UPP) to a callback function that uses your criteria for mapping actions. (Deprecated. Use `TXNActionNameMapperProcPtr` instead.)

`TXNCanRedo` (page 177) Deprecated in Mac OS X v10.4
: Returns whether the most recently undone action is redoable and indicates the type of action that can be redone. (Deprecated. Use `TXNCanRedoAction` (page 30) instead.)

`TXNCanUndo` (page 177) Deprecated in Mac OS X v10.4
: Returns whether the most recent action is undoable and provides a value that indicates the type of action than can be undone. (Deprecated. Use `TXNCanUndoAction` (page 31) instead.)

`TXNClearActionChangeCount` (page 178) Deprecated in Mac OS X v10.4
: Resets the specified action counters to zero. (Deprecated. Use `TXNClearCountForActionType` (page 32) instead.)

`TXNGetActionChangeCount` (page 179) Deprecated in Mac OS X v10.4
: Retrieves the number of times the specified action or actions have occurred since the count was initialized or cleared. (Deprecated. Use `TXNGetCountForActionType` (page 51) instead.)

`TXNSave` (page 180) Deprecated in Mac OS X v10.4
: Saves the contents of the document as the file type you specify. (Deprecated. Use `TXNWriteRangeToCFURL` (page 86) instead.)

`TXNSetDataFromCFURLRef` (page 182) Deprecated in Mac OS X v10.4
: Replaces a range of data with the contents of a file. (Deprecated. Use `TXNReadFromCFURL` (page 67) instead.)

Unsupported Functions

This section lists functions that are not supported and cannot be called in Mac OS X.

`TXNTSMCheck` (page 84)
: Checks to see if the Text Services Manager (TSM) is active.

Functions

DisposeTXNActionNameMapperUPP

Disposes of the universal procedure pointer (UPP) to your action name mapper callback function.

```
void DisposeTXNActionNameMapperUPP (
   TXNActionNameMapperUPP userUPP
);
```

Parameters

userUPP

The `TXNActionNameMapperUPP` that is to be disposed of.

Discussion

See the callback `TXNActionNameMapperProcPtr` (page 89) for more information.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`MacTextEditor.h`

DisposeTXNContextualMenuSetupUPP

Disposes of the universal procedure pointer (UPP) to your contextual menu setup callback function.

```
void DisposeTXNContextualMenuSetupUPP (
   TXNContextualMenuSetupUPP userUPP
);
```

Parameters

userUPP

The `TXNContextualMenuSetupUPP` that is to be disposed of.

Discussion

See the callback `TXNContextualMenuSetupProcPtr` (page 90) for more information.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`MacTextEditor.h`

DisposeTXNFindUPP

Disposes of the universal procedure pointer (UPP) to your find callback function.

```
void DisposeTXNFindUPP (
   TXNFindUPP userUPP
);
```

Discussion
See the callback `TXNFindProcPtr` (page 90) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

DisposeTXNScrollInfoUPP

Disposes of the universal procedure pointer (UPP) to your scrolling callback function.

```
void DisposeTXNScrollInfoUPP (
   TXNScrollInfoUPP userUPP
);
```

Discussion
See the callback `TXNScrollInfoProcPtr` (page 92) for more information.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`MacTextEditor.h`

HITextViewCopyBackgroundColor

Obtains the background color of the view.

```
OSStatus HITextViewCopyBackgroundColor (
   HIViewRef inTextView,
   CGColorRef *outColor
);
```

Parameters

inTextView
  The HITextView associated with the text object whose background color you want to copy.

outColor
  A `CGColorRef` representing the color or pattern that is used for drawing the background of the text view. If the returned `CGColorRef` is not `NULL`, it is retained on return. You are responsible for releasing this `CGColorRef` when you are no longer referencing it. If the returned value is `NULL`, the background is not drawn.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Availability
Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In
`HITextViews.h`

HITextViewCreate

Creates an HITextView that is initially invisible.

```
OSStatus HITextViewCreate (
   const HIRect *inBoundsRect,
   OptionBits inOptions,
   TXNFrameOptions inTXNFrameOptions,
   HIViewRef *outTextView
);
```

Parameters

inBoundsRect
: The bounding box of the view. Pass `NULL`, if you want to initialize the bounds of the view to `0`.

inOptions
: Reserved for future use; you must pass `0`.

inTXNFrameOptions
: The frame options you want to set for the text view.

outTextView
: On output, points to the newly-created text view.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
An HITextView is an MLTE text view that can be embedded in the HIView hierarchy. The view can be embedded in an HIScrollView if you want scroll bars and can also be used in a composited window. For more information on HIView, see the document *Introducing HIView*, available from the Apple Developer Documentation website.

When you call the function `HITextViewCreate` to create a text view, an MLTE text object (`TXNObject`) is allocated and attached to the text view. You can extract the text object by calling the function `HITextViewGetTXNObject`. You can supply the extracted text object as a parameter to many of the MLTE functions that take a text object as a parameter. However, not all MLTE functions that take a text object can operate on an MLTE object that comes from an HITextView. In general, you cannot use MLTE functions that may alter the geometry of the object or explicitly invoke drawing. If you do, the function returns the result code `kTXNDisabledFunctionalityErr`.

The following MLTE functions return an error if you pass a text object that comes from an HITextView:

- `TXNAttachObjectToWindowRef`
- `TXNGetWindowRef`
- `TXNDrawObject`
- `TXNSetScrollbarState`
- `TXNGrowWindow`
- `TXNZoomWindow`
- `TXNResizeFrame`

- `TXNSetFrameBounds`
- `TXNSetViewRect`
- `TXNDraw`
- `TXNFocus`
- `TXNUpdate`
- `TXNForceUpdate`
- `TXNPageSetup`
- `TXNPrint`
- `TXNIdle`

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HITextViews.h`

HITextViewGetTXNObject

Obtains the text object associated with an HITextView.

```
TXNObject HITextViewGetTXNObject (
   HIViewRef inTextView
);
```

Parameters

`inTextView`
The HITextView associated with the text object you want to retrieve.

Return Value
Returns the text object associated with the given view.

Discussion
You can supply the extracted text object as a parameter to many of the MLTE functions that take a text object as a parameter. However, not all MLTE functions that take a text object can operate on an MLTE object that comes from an HITextView. In general, you cannot use MLTE functions that may alter the geometry of the object or explicitly invoke drawing. If you do, the function returns the result code `kTXNDisabledFunctionalityErr`. See `HITextViewCreate` (page 22) for a list of the functions that return an error if you pass a text object that comes from an HITextView.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Related Sample Code
QTMetaData

Declared In
`HITextViews.h`

HITextViewSetBackgroundColor

Sets the background color of the view.

```
OSStatus HITextViewSetBackgroundColor (
   HIViewRef inTextView,
   CGColorRef inColor
);
```

Parameters

inTextView

The HITextView whose background color is to be set.

inColor

A `CGColorRef` representing the color or pattern that is to fill the background of the text view. The `CGColorRef` is retained by this function. If the text view already contains a background color, it is released prior to the new color being retained. If `inColor` is `NULL`, the background of the text view will not draw.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

This function allows you to provide alpha.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`HITextViews.h`

InvokeTXNActionNameMapperUPP

Calls your action name mapper callback function.

```
CFStringRef InvokeTXNActionNameMapperUPP (
   CFStringRef actionName,
   UInt32 commandID,
   void *inUserData,
   TXNActionNameMapperUPP userUPP
);
```

Parameters

actionName

The action name.

commandID

The command ID of the menu item that is to be mapped.

iUserData

A pointer to user-defined data that will be passed to your action name mapper callback.

userUPP

The callback function that is to be called. For more information, see `NewTXNActionNameMapperUPP` (page 26).

Return Value
A `CFStringRef`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`MacTextEditor.h`

InvokeTXNContextualMenuSetupUPP

Calls your contextual menu setup callback function.

```
void InvokeTXNContextualMenuSetupUPP (
   MenuRef iContextualMenu,
   TXNObject object,
   void *inUserData,
   TXNContextualMenuSetupUPP userUPP
);
```

Parameters

`iContextualMenu`
The contextual menu.

`object`
The TXNObject for which the contextual menu is to be displayed.

`iUserData`
A pointer to user-defined data that will be passed to your contextual menu setup callback.

`userUPP`
The callback function that is to be called. For more information, see `NewTXNContextualMenuSetupUPP` (page 27).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`MacTextEditor.h`

InvokeTXNFindUPP

Calls your find callback function.

```
OSStatus InvokeTXNFindUPP (
   const TXNMatchTextRecord *matchData,
   TXNDataType iDataType,
   TXNMatchOptions iMatchOptions,
   const void *iSearchTextPtr,
   TextEncoding encoding,
   TXNOffset absStartOffset,
   ByteCount searchTextLength,
   TXNOffset *oStartMatch,
   TXNOffset *oEndMatch,
   Boolean *ofound,
   URefCon refCon,
   TXNFindUPP userUPP
);
```

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
You can call an invoke function rather than calling your routine directly if you want to support code portability across compiler targets. See the callback `TXNFindProcPtr` (page 90) for parameter descriptions and other information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

InvokeTXNScrollInfoUPP

Calls your scrolling callback function.

```
void InvokeTXNScrollInfoUPP (
   SInt32 iValue,
   SInt32 iMaximumValue,
   TXNScrollBarOrientation iScrollBarOrientation,
   SRefCon iRefCon,
   TXNScrollInfoUPP userUPP
);
```

Discussion
You can call an invoke function rather than calling your routine directly if you want to support code portability across compiler targets. See the callback `TXNScrollInfoProcPtr` (page 92) for parameter descriptions and other information.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`MacTextEditor.h`

NewTXNActionNameMapperUPP

Creates a new universal procedure pointer (UPP) to an action name mapper callback function.

```
TXNActionNameMapperUPP NewTXNActionNameMapperUPP (
   TXNActionNameMapperProcPtr userRoutine
);
```

Parameters

userRoutine

The action name mapper callback function for which a UPP is to be created.

Return Value

A universal procedure pointer.

Discussion

See the callback `TXNActionNameMapperProcPtr` (page 89) for more information.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`MacTextEditor.h`

NewTXNContextualMenuSetupUPP

Creates a new universal procedure pointer (UPP) to a contextual menu setup callback function.

```
TXNContextualMenuSetupUPP NewTXNContextualMenuSetupUPP (
   TXNContextualMenuSetupProcPtr userRoutine
);
```

Parameters

userRoutine

The contextual menu setup callback function for which a UPP is to be created.

Return Value

A universal procedure pointer.

Discussion

For more information, see the callback `TXNContextualMenuSetupProcPtr` (page 90).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`MacTextEditor.h`

NewTXNFindUPP

Creates a new universal procedure pointer (UPP) to a find callback function that uses your criteria for matching.

```
TXNFindUPP NewTXNFindUPP (
   TXNFindProcPtr userRoutine
);
```

Return Value

A universal procedure pointer.

Discussion
See the callback `TXNFindProcPtr` (page 90) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

NewTXNScrollInfoUPP

Creates a new universal procedure pointer (UPP) to a scrolling callback function.

```
TXNScrollInfoUPP NewTXNScrollInfoUPP (
   TXNScrollInfoProcPtr userRoutine
);
```

Return Value
A universal procedure pointer.

Discussion
See the callback `TXNScrollInfoProcPtr` (page 92) for more information.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`MacTextEditor.h`

TXNAdjustCursor

Obtains the current cursor position and draws the cursor in a form appropriate to the content over which it is placed.

```
void TXNAdjustCursor (
   TXNObject iTXNObject,
   RgnHandle ioCursorRgn
);
```

Parameters

iTXNObject
The text object that identifies the text area for which MLTE should adjust the cursor.

ioCursorRgn
A handle to a region created by your application. Pass `NULL` if you do not want `TXNAdjustCursor` to provide information about the cursor position to your application. If you do want to obtain the cursor's current position, pass a valid region handle in this parameter. If you pass a valid region handle and the cursor is over the text area or its scroll bars, on return `TXNAdjustCursor` sets the region to a 2-pixel by 2-pixel square, centered on the cursor's hot spot. If the cursor is not over the text area or its scroll bars, or if you have passed `NULL`, `TXNAdjustCursor` does not adjust the input value.

Discussion
If the cursor is over a text area, `TXNAdjustCursor` sets the cursor to an I-beam. If the cursor is over graphics, a sound file, a movie, a scroll bar, or outside of a window, `TXNAdjustCursor` sets the cursor to an arrow. Before you call the `TXNAdjustCursor` function, you should make sure that the window belongs to your application.

You can pass the region handle returned by the `TXNAdjustCursor` function in the `ioCursorRgn` parameter to the `WaitNextEvent` function; this ensures that you receive mouse-moved events if the cursor moves outside that region. If you then receive a mouse-moved event, you can call `TXNAdjustCursor` again to ensure that the cursor type is appropriate to its new position. Alternately, to ensure that the cursor is adjusted correctly, you can simply call `TXNAdjustCursor` with every event that your application receives.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNAttachObjectToWindowRef

Attaches a text object to a window.

```
OSStatus TXNAttachObjectToWindowRef (
   TXNObject iTXNObject,
   WindowRef iWindowRef
);
```

Parameters

`iTXNObject`
The text object with which you want to associate the window. You can call the function `TXNCreateObject` to allocate a text object.

`iWindowRef`
A `WindowRef` that specifies the window to which you want to attach the text object.

Return Value
A result code. See "MLTE Result Codes" (page 160). Returns `paramErr` if the text object that you pass is invalid.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNBeginActionGroup

Starts an action group.

```
OSStatus TXNBeginActionGroup (
   TXNObject iTXNObject,
   CFStringRef iActionGroupName
);
```

Parameters

iTXNObject

The text object for which an action group is to be started.

iActionGroupName

A client-supplied string that is to be used to describe the action group.

Return Value

A result code. See "MLTE Result Codes" (page 160). The error `kTXNOperationNotAllowedErr` is returned if an undo action group has already been started but has not yet terminated.

Discussion

Every supported edit action after `TXNBeginActionGroup` is called is added to the group until`TXNCanUndoAction` (page 31) is called. When MLTE receives an undo or redo command, it treats all actions added to the group as a single operation to undo or redo. Nesting of groups is not allowed. Calling `TXNBeginActionGroup` twice without calling `TXNCanUndoAction` in between results in an error. If an action group is active, `TXNCanUndoAction` (page 31) and `TXNCanRedoAction` (page 30) return `false`.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNCanRedoAction

Indicates whether an action can be redone.

```
Boolean TXNCanRedoAction (
   TXNObject iTXNObject,
   CFStringRef *oActionName
);
```

Parameters

iTXNObject

The text object having an action that is to be queried.

oActionName

On input, a pointer a `CFStringRef` that, on return, contains the name of the action that can be redone, if there is one. The returned string is either a string defined by MLTE or the string that you passed to `TXNBeginActionGroup` (page 29) to create a new action group. You are responsible for retaining and releasing the string. Pass `NULL` if you don't want to receive the name of the action.

Return Value

A Boolean whose value is `true` if the last action can be redone; you should enable the Redo item in the Edit menu, if there is one. If this function returns `false`, the last action cannot be redone and you should not enable the Redo item in the Edit menu.

Discussion
This function tells the client whether the current item on the undo stack is redoable and is usually used to determine whether the Redo item in the Edit menu should be enabled. This function optionally obtains the action name that should be used in the Redo item. When the current undo item is an action group, the string used to name the group is returned. For information on action groups, see `TXNBeginActionGroup` (page 29) and `TXNEndActionGroup` (page 44).

Availability
Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNCanUndoAction

Indicates whether an action can be undone.

```
Boolean TXNCanUndoAction (
   TXNObject iTXNObject,
   CFStringRef *oActionName
);
```

Parameters

iTXNObject

The text object having an action that is to be queried.

oActionName

On input, a pointer a `CFStringRef` that, on return contains the name of the action that can be undone, if there is one. The returned string is either a string defined by MLTE or the string that you passed to `TXNBeginActionGroup` (page 29) to create a new action group. You are responsible for retaining and releasing the string. Pass `NULL` if you don't want to receive the name of the action.

Return Value
A Boolean whose value is `true` if the last action can be undone; you should enable the Undo item in the Edit menu, if there is one. If this function returns `false`, the last action cannot be undone and you should not enable the Undo item in the Edit menu.

Discussion
This function is usually used to determine whether the Undo item in the Edit menu should be enabled and to obtain the action name that should be used in that item. When the last action is an action group, the string used to name the group is returned.

If you have asked MLTE to handle updating for the Redo and Undo commands in the Edit menu, you should call `TXNSetActionNameMapper` (page 72) after calling `TXNCanUndoAction` so that MLTE can call back to get the correct strings for those items.

Availability
Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNClear

Deletes the current selection.

```
OSStatus TXNClear (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object for the current text area.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

To remove a selected object from a text area, a user can either press the Delete key or choose Clear from the Edit menu. Before you call the `TXNClear` function, you can use the `TXNIsSelectionEmpty` (page 63) function to determine whether any text is selected. Unlike the function `TXNCut` (page 37), the `TXNClear` function does not add the deleted selection to the private MLTE scrap.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNClearCountForActionType

Sets the counter for the specified action type to zero.

```
OSStatus TXNClearCountForActionType (
   TXNObject iTXNObject,
   CFStringRef iActionTypeName
);
```

Parameters

iTXNObject

The text object having one or more counters that are to be set to zero.

iActionTypeName

The action type for which the counter is to be set to zero. This parameter can be a string that was passed to `TXNBeginActionGroup` (page 29) or one of the constants described inAction Constants (page 106).

Return Value

A result code. See "MLTE Result Codes" (page 160).

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNClick

Processes a mouse-down event in a window's content area.

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

Parameters

`iTXNObject`

The text object in which the mouse-down event occurred.

`iEvent`

A pointer to the event record that contains the mouse-down event to process.

Discussion

When you pass the event to the `TXNClick` function, it responds to the user's action by scrolling, selecting text, playing a sound or movie, handling a drag–and-drop operation, or responding to a double- or triple-click, as appropriate. Before you call `TXNClick`, you should make sure that the front window belongs to your application.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNCopy

Copies the current selection to the private MLTE scrap.

```
OSStatus TXNCopy (
   TXNObject iTXNObject
);
```

Parameters

`iTXNObject`

The text object for the current text area.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

You can use the `TXNCopy` function to respond to a user-requested copy action. Before you call `TXNCopy`, you can use the `TXNIsSelectionEmpty` (page 63) function to determine whether any text is selected.

The `TXNCopy` function copies the current selection to the MLTE scrap. In a Carbon application, the Scrap Manager automatically converts your application's private scrap to the Clipboard so it is available to other applications. In a Classic application, you must call the function `TXNConvertToPublicScrap` (page 166) after you call `TXNCopy`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNCopyTypeIdentifiersForRange

Obtains an array of universal type identifiers for a TXNObject.

```
OSStatus TXNCopyTypeIdentifiersForRange (
   TXNObject iTXNObject,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset,
   CFArrayRef *oTypeIdentifiersForRange
);
```

Parameters

`iTXNObject`
: The text object.

`iStartOffset`
: The starting offset in `iTXNObject`.

`iEndOffset`
: The ending offset in `iTXNObject`.

`oTypeIdentifiersForRange`
: A pointer to a `CFArrayRef`. On return, the array contains the list of universal type identifiers that MLTE supports. Each entry in the array is a `CFStringRef`.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

Some file formats support limited embedding of data when writing to disk, and use attachments, such as Rich Text Format (RTF), instead.

Use this function to get a list of universal type identifiers that can be used when calling `TXNWriteRangeToCFURL` (page 86) to write the object out to disk with no data loss. Note that support for new document formats could be added in the future.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNCountRunsInRange

Obtains a count of the style runs in a range of data.

```
OSStatus TXNCountRunsInRange (
   TXNObject iTXNObject,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset,
   ItemCount *oRunCount
);
```

Parameters

iTXNObject
The text object for the current text area.

iStartOffset
The beginning offset of the range of data in the document that you want to examine. Note that this offset is a generic counter of elements (such as characters, pictures, and movies), not an offset into memory.

iEndOffset
The ending offset of the range of data in the document that you want to examine. Note that this offset is a generic counter of elements, not an offset into memory.

oRunCount
On return, a pointer to the number of style runs in the specified range.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
Given a range of data in a document that is specified by a starting and ending offset, you can use the `TXNCountRunsInRange` function to obtain a count of the changes in text styles, graphics, movies, or sounds in that range. Once you have a run count, you can supply this information to the function `TXNGetIndexedRunInfoFromRange` (page 55) in order to obtain information about the runs themselves.

Offsets in MLTE are always character offsets.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNCreateObject

Creates a new MLTE text object which is an opaque structure that handles text formatting at the document level.

```
OSStatus TXNCreateObject (
   const HIRect *iFrameRect,
   TXNFrameOptions iFrameOptions,
   TXNObject *oTXNObject
);
```

Parameters

iFrameRect

A pointer to a rectangle used to specify the destination and view rectangles for the text object. Pass `NULL` if you want to use the window port rectangle as the view and destination rectangles when the object is attached later to a window. See the function `TXNAttachObjectToWindowRef`.

iFrameOptions

A value that specifies the options you want the frame to support. See Frame Option Masks (page 136) for a description of possible values.

oTXNObject

On input, a pointer to a structure of type `TXNObject`. On output, points to the opaque text object data structure allocated by the function. You need to pass this object to most MLTE functions.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

For each document, a new text object is allocated by the `TXNCreateObject` function and returned in the `oTXNObject` parameter. The object is allocated only if no errors occur. If there is an error during the allocation process, MLTE frees the text object.

If you are writing a text editing application, you may want to call the `TXNCreateObject` function when the application launches (a new document will be displayed) and whenever the user selects New from the File menu. In addition, many MLTE functions require you to pass a text object.

If you want to create a read-only document, you need to pass the option `kTXNReadOnlyMask` in the `iFrameOptions` parameter. Note that this option puts the text object into a state that does not allow user input. However, your application can put data into the text object by calling the function `TXNSetData`. If you want the text object set into a more restrictive read-only state that does not allow user input or your application to put data into the text object programmatically, you need to call the function `TXNSetTXNObjectControls`, passing the tag `kTXNIOPrivilegesTag`. If you choose to set the text object into this restrictive state, you will get an error if you try to call the function `TXNSetData` on the text object. (In this case, you can change the text object to a less restrictive state by calling `TXNSetTXNObjectControls`, passing the tag `kTXNNoUserIOTag`.)

Because of how MLTE uses Carbon events internally, the window in which the document is displayed must have the standard event handlers installed. You can install standard event handlers in one of the following ways:

- When you create the window, add the attribute `kWindowStandardHandlerAttribute` to the window. See *Window Manager Reference* for more information.
- Call the Carbon Event Manager function `InstallStandardEventHandler` on the window's event target. See *Handling Carbon Events* for more information.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNCut

Deletes the current selection and copies it to the private MLTE scrap.

```
OSStatus TXNCut (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object for the current text area.

Return Value
A result code. See "MLTE Result Codes" (page 160).`TXNCut` also returns Scrap Manager errors.

Discussion
You can use the `TXNCut` function to respond to a user-requested cut action. Before you call `TXNCut`, you can use the `TXNIsSelectionEmpty` (page 63) function to determine whether any text is selected. The `TXNCut` function deletes the current selection and then copies it to the private MLTE scrap. In a Carbon application, the Scrap Manager automatically converts your application's private scrap to the Clipboard so it is available to other applications. In a Classic application, you must call the function `TXNConvertToPublicScrap` (page 166) after you call `TXNCut` to move the selection to the Clipboard.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNDataSize

Reports the amount of memory used to hold the text in a given text object.

```
ByteCount TXNDataSize (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object that you want to examine.

Return Value
The number of bytes required to hold the characters.

Discussion
You can use this function to determine how large a handle should be if, for example, you copy text. Note that because every individual sound, picture, or movie in a text object is represented by a single character in the text buffer, the `TXNDataSize` function returns a value that does not necessarily represent the true size of any non-text data.

If you are using Unicode and you want to know the number of characters, divide the returned `ByteCount` value by `sizeof(UniChar)` or 2, since MLTE uses the 16-bit Unicode Transformation Format (UTF-16).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNDeleteObject

Deletes a previously allocated text object.

```
void TXNDeleteObject (
   TXNObject iTXNObject
);
```

Parameters

`iTXNObject`
: The text object you want to delete.

Discussion
You should call the `TXNDeleteObject` function when you close the window associated with a text object. The function `TXNDeleteObject` releases the specified text object and all associated data structures from memory. If the object has multiple frames, all frames are deleted.

Version Notes
Multiple frames are not yet implemented in MLTE.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNDragReceiver

Handles dragged data in a text object for which a custom drag handler is already in place.

```
OSErr TXNDragReceiver (
   TXNObject iTXNObject,
   TXNFrameID iTXNFrameID,
   WindowRef iWindow,
   DragReference iDragReference,
   Boolean iDifferentObjectSameWindow
);
```

Parameters

`iTXNObject`
: The text object that is receiving the dragged data.

iTXNFrameID
: The frame ID of the text object that is receiving the dragged data. You obtain a `TXNFrameID` when you create a text object with the `TXNNewObject` (page 170) function.

iWindow
: A pointer to the window containing the text object that is receiving the dragged data. You obtain this pointer from the appropriate Drag Manager function.

iDragReference
: The drag reference you want MLTE to handle. You obtain a drag reference by calling the appropriate Drag Manager function.

iDifferentObjectSameWindow
: A `Boolean` value. Pass `true` if the drag operation is in the same window that it started in, but in a different text object within that window. If there is only one text object in a window, you should always pass `false`. You should also pass `false` if the drag operation has moved into a different window than the one in which it originated. You can determine whether the drag operation has left the originating window by calling the Drag Manager function `GetDragAttributes`.

Return Value
A result code. See "MLTE Result Codes" (page 160). A Drag Manager result code.

Discussion
You would not typically use the `TXNDragReceiver` function, because MLTE provides basic drag management for you.

However, you might call `TXNDragReceiver` if your application needs to examine the dragged data prior to MLTE handling it, or if you have multiple text objects in a window, or if you have your own drag management infrastructure that you want to use.

You must inform MLTE that you wish to handle some aspect of the drag process by passing the `TXNFrameOptions` value `kTXNDoNotInstallDragProcsMask` in the `iFrameOptions` parameter of `TXNNewObject` (page 170). If you do so, you are responsible for calling the drag handlers for the drag operation. Then, you should call `TXNDragReceiver` when your drag receiver is called and you want MLTE to take over control of the drag reception process.

When you call `TXNDragReceiver`, MLTE takes over the drag operation and handles everything from that point onward. This includes determining whether the text object is a valid drop target and if the dragged data is an MLTE-supported type, as well as managing the addition of the dragged data to the text object.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNDragTracker

Handles tracking a drag event in a text object for which a custom drag handler is already in place.

```
OSErr TXNDragTracker (
   TXNObject iTXNObject,
   TXNFrameID iTXNFrameID,
   DragTrackingMessage iMessage,
   WindowRef iWindow,
   DragReference iDragReference,
   Boolean iDifferentObjectSameWindow
);
```

Parameters

iTXNObject
: The text object in which you need to track a drag event.

iTXNFrameID
: The frame ID of the text object in which you need to track a drag event. You obtain a `TXNFrameID` when you create a text object with the `TXNNewObject` (page 170) function.

iMessage
: A drag message obtained from the appropriate Drag Manager function.

iWindow
: A pointer to the window containing the text object in which you need to track a drag event. You obtain this pointer from the appropriate Drag Manager function.

iDragReference
: The drag reference that you want MLTE to handle. You obtain a drag reference by calling the appropriate Drag Manager function.

iDifferentObjectSameWindow
: A value of type `Boolean`. If your application displays more than one text object per window, pass `true` when the drag operation moves out of one object's view rectangle and into another text object's view rectangle.

Return Value
A result code. See "MLTE Result Codes" (page 160). A Drag Manager result code.

Discussion
You would not typically use the `TXNDragTracker` function, because MLTE provides basic drag management for you.

However, you might call `TXNDragTracker` if your application needs to examine the dragged data prior to MLTE handling it, or if you have multiple text objects in a window, or if you have your own drag management infrastructure that you want to use.

You must inform MLTE that you wish to handle some aspect of the drag process by passing the `TXNFrameOptions` value `kTXNDoNotInstallDragProcsMask` in the `iFrameOptions` parameter of `TXNNewObject` (page 170). If you do so, you are responsible for calling the drag handlers for the drag operation. Then, you should call `TXNDragTracker` when your drag tracker is called and you want MLTE to take over control of the drag tracking process.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNDrawCFStringTextBox

Draws text from a Core Foundation string (`CFString`) in the specified rectangle.

```
OSStatus TXNDrawCFStringTextBox (
   CFStringRef iText,
   Rect *ioBox,
   ATSUStyle iStyle,
   const TXNTextBoxOptionsData *iOptions
);
```

Parameters

`iText`

A reference to the Core Foundation string you want drawn in the text box. A Core Foundation string is an array of Unicode characters along with a count of the number of characters in the string. A CFString object is stored as efficiently as possible, so the memory required to store the string is often less than that required to store a simple array of Unicode characters.

`ioBox`

On input, a pointer to the rectangle that specifies the text box in which the text is to be displayed. On return, MLTE updates the rectangle to reflect the minimum bounding rectangle that encloses the text. If you pass the constant `kTXNDontUpdateBoxRectMask` in the `ioOptions` parameter then the rectangle is not updated. If the rectangle already has text displayed in it, you should call the QuickDraw function `EraseRect` before you call the `TXNDrawUnicodeTextBox` function. The drawing is clipped to the rectangle unless you specify a rotation as one of the text options you set with the `ioOptions` parameter.

`iStyle`

An ATSUI style to use to display the text. This parameter is optional. If you pass `NULL`, MLTE creates an ATSUI style based on the information (text size, type face, color, and so forth) associated with the current graphics port. See the Apple Type Services for Unicode Imaging documentation for a description of the `ATSUStyle` data type.

`iOptions`

A pointer to a `TXNTextBoxOptionsData` structure associated with this text object. This is optional. If you pass `NULL`, MLTE uses the settings for the current graphics port. You can use the `TXNTextBoxOptionsData` structure to specify a number of options, such as text orientation (vertical or horizontal), justification, alignment, and font substitution for text that cannot be rendered using the specified font. See Text Box Options Masks (page 157) for a description of the options you can specify.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

You can use the `TXNDrawCFStringTextBox` function to display mono-style Unicode text. You do not need to initialize MLTE to use this function because it uses Apple Type Services for Unicode Imaging (ATSUI) directly.

If you display text justified, it is justified in the direction of the display. Horizontal text is justified horizontally, but not vertically. Vertical text is justified vertically, but not horizontally.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNDrawObject

Draws a text object in the last window set by your application.

```
OSStatus TXNDrawObject (
   TXNObject iTXNObject,
   const HIRect *iClipRect,
   TXNDrawItems iDrawItems
);
```

Parameters

iTXNObject
: The text object you want to draw. You can call the function `TXNCreateObject` to allocate a text object.

iClipRect
: A pointer to an `HIRect` data structure. If the rectangle is `NULL` MLTE uses the view rectangle when drawing. Otherwise, MLTE uses the rectangle that is the intersection of the `iClipRect` rectangle and the view rectangle.

iDrawItems
: A Draw Items Masks (page 124) value that specifies which elements are drawn. Pass `kTXNDrawItemScrollbarsMask` if you want the scroll bars drawn; `kTXNDrawItemTextMask` to render the text; `kTXNDrawItemTextAndSelectionMask` to render the text and the current selection, and `kTXNDrawItemAllMask` to draw the scroll bars, text, and the current selection.

Discussion

This function has no effect for text objects that have the visibility tag set to `false`.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNDrawUnicodeTextBox

Draws a Unicode string in the specified rectangle.

```
OSStatus TXNDrawUnicodeTextBox (
   const UniChar iText[],
   UniCharCount iLen,
   Rect *ioBox,
   ATSUStyle iStyle,
   const TXNTextBoxOptionsData *iOptions
);
```

Parameters

iText
: The Unicode string you want drawn in the text box. The string should be in 16-bit Unicode Transformation Format (UTF-16).

iLen
: The number of Unicode characters contained in the Unicode string.

ioBox

On input, a pointer to the rectangle, in local coordinates, that specifies the text box in which the text is to be displayed. On return, MLTE updates the rectangle to reflect the minimum bounding rectangle that encloses the text. If you pass the constant `kTXNDontUpdateBoxRectMask` in the `ioOptions` parameter, the rectangle is not updated. If the rectangle already has text displayed in it, you should call the QuickDraw function `EraseRect` before you call the `TXNDrawUnicodeTextBox` function. The drawing is clipped to the rectangle unless you specify a rotation as one of the text options you set with the `ioOptions` parameter.

iStyle

An ATSUI style to use to display the text. This parameter is optional. If you pass `NULL`, MLTE creates an ATSUI style based on the information (text size, type face, color, and so forth) associated with the current graphics port. See the Apple Type Services for Unicode Imaging documentation for a description of the `ATSUStyle` data type.

ioOptions

A pointer to a `TXNTextBoxOptionsData` structure associated with this text object. This is optional. If you pass `NULL`, MLTE uses the settings for the current graphics port. You can use the `TXNTextBoxOptionsData` structure to specify a number of options, such as text orientation (vertical or horizontal), justification, alignment, and font substitution for text that cannot be rendered using the specified font. See Text Box Options Masks (page 157) for a description of the options you can specify.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

You can use the `TXNDrawUnicodeTextBox` function to display mono-style Unicode text. You do not need to initialize MLTE to use this function because it uses Apple Type Services for Unicode Imaging (ATSUI) directly.

If you display text justified, it is justified in the direction of the display. Horizontal text is justified horizontally, but not vertically. Vertical text is justified vertically, but not horizontally.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNEchoMode

Determines whether a specified character is drawn instead of the glyph associated with the input character.

```
OSStatus TXNEchoMode (
   TXNObject iTXNObject,
   UniChar iEchoCharacter,
   TextEncoding iEncoding,
   Boolean iOn
);
```

Parameters

iTXNObject

The text object for the current text area.

`iEchoCharacter`
: A value that specifies the substitute character.

`iEncoding`
: The text encoding from which the substitute character is drawn. See the Text Encoding Conversion Manager reference documentation for a discussion of the text encoding data type and of possible text encoding values.

`iOn`
: A `Boolean` value. Pass `true` to turn on character substitution. Pass `false` to turn it off. When you enable character substitution for a text object, all characters in the text area have the character specified by the `iEchoCharacter` parameter substituted for the actual glyph when MLTE draws the text.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
You can use the `TXNEchoMode` function when you want to hide what the user types, such as a password in a login dialog box.

The substitution character is a `UniChar` data type to facilitate passing any 2-byte character. The encoding parameter actually determines the encoding MLTE uses to locate a font and display a character. Thus if you want to display the diamond character from the Shift-JIS encoding for Mac OS, you would pass the value `0x86A6` for the character, but pass an encoding value that represents Mac OS Japanese encoding.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNEndActionGroup

Ends an action group.

```
OSStatus TXNEndActionGroup (
   TXNObject iTXNObject
);
```

Parameters

`iTXNObject`
: The text object for which an action group is to be ended.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
The call is ignored if there is no active action group.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNFind

Finds a piece of text or a graphics, sound, or movie object.

```
OSStatus TXNFind (
   TXNObject iTXNObject,
   const TXNMatchTextRecord *iMatchTextDataPtr,
   TXNDataType iDataType,
   TXNMatchOptions iMatchOptions,
   TXNOffset iStartSearchOffset,
   TXNOffset iEndSearchOffset,
   TXNFindUPP iFindProc,
   SRefCon iRefCon,
   TXNOffset *oStartMatchOffset,
   TXNOffset *oEndMatchOffset
);
```

Parameters

iTXNObject
: The text object to be searched.

iMatchTextDataPtr
: A pointer to a data structure that contains the text to match, the length of that text, and the text's encoding. Pass `NULL` if you are looking for a graphics, sound, or movie object.

iDataType
: The type of data for which you want to search. See Supported Data Types (page 152) for a description of possible values. If the data type is `kTXNPictureFile`, `kTXNMovieFile`, or `kTXNSoundFile`, the default behavior is to match on any nontext object. If you want to find a specific data type, you can provide a custom find callback or ignore types that do not match what you want to find.

iMatchOptions
: The matching rules to use in the find operation. See Search Criteria Masks (page 149) for a description of possible values.

iStartSearchOffset
: The offset at which the search should begin. You can use `kTXNStartOffset` if you want to search from the start of the object's data.

iEndSearchOffset
: The offset at which the search should end. You can use `kTXNEndOffset` if you want to search to the end of the object's data.

iFindProc
: The custom callback you want used in place of the default matching behavior. You can pass `NULL` if you want to use the default matching behavior.

iRefCon
: An signed 32-bit integer. You can use this for whatever your application needs. It is passed to the custom callback specified by `iFindProc`.

oStartMatchOffset
: On return, a pointer to the absolute offset that identifies the start of the match. It is set to a value of `kTXNUseCurrentSelection` if there is no match.

oEndMatchOffset
: On return, a pointer to the absolute offset that identifies the end of the match. It is set to a value of `kTXNUseCurrentSelection` if there is no match.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
By default, text is matched on the basis of a binary comparison. If you set the `iMatchOptions` variable to ignore case, the characters to be searched are duplicated and case neutralized. If the want to search a large amount of text, a case insensitive search can fail due to insufficient memory.

If you set the `iMatchOptions` variable to find an entire word, then once a match is found, the matched text is tested to see if it is a word. If the `kTXNUseEncodingWordRulesBit` is set, then the Script Manager `FindWord` function is called to make this determination. If the text being searched is Unicode text, then the ATSUI word-determining functions are used to test for a word.

If the application is looking for a nontext type, then each nontext type in the document is returned. The `iFindProc` parameter lets you provide a more elaborate search engine (such as a regular expression processor) should you need one.

Offsets in MLTE are always character offsets.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNFlattenObjectToCFDataRef

Flattens a text object so it can be saved to disk or embedded with other data.

```
OSStatus TXNFlattenObjectToCFDataRef (
   TXNObject iTXNObject,
   TXNDataType iTXNDataType,
   CFDataRef *oDataRef
);
```

Parameters

iTXNObject
: The text object that identifies the document you want to flatten. You can either call the function `TXNCreateObject` to allocate a text object or you can call the function `HITextViewGetTXNObject` (page 23) to obtain the text object associated with an HITextView.

iTXNDataType
: A value that specifies the format in which the data is written out.

oDataRef
: On input, points to a structure of type `CFDataRef`. On output, points to a flattened version of the text object in the format specified by the `iTXNDataType` parameter. You are responsible to retain the returned `CFDataRef`.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
This function supports the following data types:

- `kTXNTextData`, for text data.
- `kTXNUnicodeTextData`, for plain UTF-16.
- `kTXNTextAndMultimediaData`, for data in MLTE format.
- `kTXNRichTextFormatData`, for data in RTF format.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNFocus

Changes the focus of a text object.

```
void TXNFocus (
   TXNObject iTXNObject,
   Boolean iBecomingFocused
);
```

Parameters

iTXNObject
The text object whose focus you want to change.

iBecomingFocused
If you pass `true`, the text object receives focus. This means the current selection or insertion point is active, text input appears at the insertion point, and the keyboard and font are synchronized. (Note that the font and keyboard are synchronized only if keyboard synchronization is enabled. See Keyboard Synchronization Settings (page 143).) If the scroll bars are not already active, they are activated. If you pass `false`, the text object's current selection or insertion point is inactive.

Discussion
You should use the `TXNActivate` (page 164) function to make scroll bars active while text input is not focused. This behavior is often desirable for windows with multiple text areas that are scrollable.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNForceUpdate

Forces an update of the view rectangle and the scroll bars.

```
void TXNForceUpdate (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object you want to update.

Discussion

This function operates similarly to the Window Manager functions `InvalRect` and `InvalRgn`. For example, when the user increases the size of a window that contains text from a text object, the `TXNForceUpdate` function adds the new region (including two rectangles formerly occupied by the scroll bars in the smaller content area) to the update region.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNGetAccessibilityHIObject

Obtains an `HIObjectRef` representing the MLTE object for accessibility purposes.

```
OSStatus TXNGetAccessibilityHIObject (
   TXNObject iTXNObject,
   HIObjectRef *oHIObjectRef
);
```

Parameters

iTXNObject

The text object.

oHIObjectRef

On input, a pointer an `HIObjectRef` that, on return, represents the text object specified by `iTXNObject` as an accessible object.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

For each MLTE object that a view creates, the view needs to call this function to get an `HIObjectRef` that can be used to represent the MLTE object as an accessible object.

After the view gets this `HIObjectRef`, it must add the `HIObjectRef` as a child of itself. The accessibility engine then routes events to MLTE accessible objects automatically.

The view must install Carbon Event handlers for `kEventAccessibleGetAllAttributeNames` and `kEventAccessibleGetNamedAttribute`, using the `HIObjectRef` as the target, to provide information for at least the following required attributes:

- kAXRoleAttribute
- kAXRoleDescriptionAttribute
- kAXWindowAttribute

- kAXPositionAttribute
- kAXSizeAttribute

MLTE also installs handlers for `kEventAccessibleGetAllAttributeNames` and `kEventAccessibleGetNamedAttribute`. Note that these handlers are not called unless the client-installed handlers return `eventNotHandledErr`. These handlers return information for the following attributes:

- kAXEnabledAttribute
- kAXFocusedAttribute
- kAXValueAttribute
- kAXSelectedTextAttribute
- kAXSelectedTextRangeAttribute
- kAXNumberOfCharactersAttribute
- kAXLineForIndexParameterizedAttribute
- kAXRangeForLineParameterizedAttribute
- kAXStringForRangeParameterizedAttribute
- kAXRangeForPositionParameterizedAttribute
- kAXRangeForIndexParameterizedAttribute
- kAXBoundsForRangeParameterizedAttribute
- kAXStyleRangeForIndexParameterizedAttribute

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetChangeCount

Retrieves the number of times a document has been changed.

```
ItemCount TXNGetChangeCount (
   TXNObject iTXNObject
);
```

Parameters

`iTXNObject`
The text object whose changes you want to count.

Return Value
The total number of changes since the document was last saved. If the document is new, the number of changes since it was created.

Discussion
The change count increments for every executed command such as Cut or Copy. An uninterrupted sequence of key-down events increments the count by 1. The count is cleared each time the text object is saved. You can use this function to determine if your application should make the Save item in the File menu active.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetCommandEventSupport

Obtains the command event support that is currently set for an MLTE object.

```
OSStatus TXNGetCommandEventSupport (
   TXNObject iTXNObject,
   TXNCommandEventSupportOptions *oOptions
);
```

Parameters

iTXNObject
The text object.

oOptions
A pointer to a value of type `TXNCommandEventSupportOptions` that, on return, contains the option settings for the text object specified by `iTXNObject`. For possible values, see Command Event Support Options (page 116).

Return Value
A result code. See "MLTE Result Codes" (page 160).

Availability
Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetContinuousTypeAttributes

Checks to see if the attributes of the current selection are continuous.

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

Parameters

iTXNObject
The text object that contains the current selection.

oContinuousFlags
: On return, a pointer to a value that specifies whether text attributes are continuous. See Continuous Style Information Masks (page 118) for a description of possible values. If a particular bit is set and if your application has passed a `tag` value in a `TXNTypeAttributes` structure in the array that corresponds to the bit, then your application should display a check mark next to the appropriate menu item.

ioCount
: The number of `TXNTypeAttributes` structures in the `ioTypeAttributes` array.

ioTypeAttributes
: An array of `TXNTypeAttributes` structures. The `tag` values in this array indicate the text attributes in which the application is interested. It cannot be `NULL`. If you are using ATSUI and you want to know the ATSUI font ID, you should set the `tag` field to `kATSUFontTag`, which is a constant in `ATSUnicode.h`. If you are using ATSUI and you set the `tag` field to a QuickDraw font ID, then the QuickDraw font to which the ATSUI font maps is returned. Note that this is not always the same font since many ATSUI fonts are not supported by QuickDraw.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
You can use this function to determine whether your application should display check marks in the Font, Style, Size, and Color menus. If these are the only attributes in which you are interested, you can use this function on systems that use QuickDraw or Apple Type Services for Unicode Imaging (ATSUI).

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetCountForActionType

Gets the number of times a given type of action has occurred.

```
OSStatus TXNGetCountForActionType (
   TXNObject iTXNObject,
   CFStringRef iActionTypeName,
   ItemCount *oCount
);
```

Parameters

iTXNObject
: The text object to query.

iActionTypeName
: The action type that is to be included when retrieving the count. This parameter can be the string that was passed to `TXNBeginActionGroup` (page 29) or one of the constants described in Action Constants (page 106).

oCount
: On return, the number of times the action specified by `iActionTypeName` has occurred.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
Call `TXNClearCountForActionType` (page 32) to reset the counters.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetData

Copies a range of data.

```
OSStatus TXNGetData (
   TXNObject iTXNObject,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset,
   Handle *oDataHandle
);
```

Parameters

`iTXNObject`
The text object that contains the data you want to copy.

`iStartOffset`
The absolute offset from which data copying should begin. Make sure the `iStartOffset` and `iEndOffset` parameters do not specify a range that includes text and nontext data (that is, crosses a data type boundary). You can use the `TXNGetSelection` (page 57) function to get the absolute offsets of the current selection.

`iEndOffset`
The absolute offset at which data copying should end. You can use the `TXNGetSelection` function to get the absolute offsets of the current selection.

`oDataHandle`
A pointer to a handle. On return, the handle points to the requested data. `TXNGetData` allocates the handle as necessary. Your application must dispose of the handle.

Return Value
A result code. See "MLTE Result Codes" (page 160). The `iStartOffset` and `iEndOffset` parameters can specify a range that crosses a style run boundary but not a range that crosses a data type boundary. If the range includes text and nontext data, the `TXNGetData` function returns the error code `kTXNIllegalToCrossDataBoundariesErr`.

Discussion
You first need to use the `TXNCountRunsInRange` (page 34) function to find the number of data runs in a given range. Then you can examine each run's type and text attributes with the `TXNGetIndexedRunInfoFromRange` (page 55) function. Finally, use the `TXNGetData` function to examine data for each run of interest to you. The function does not check to see that the copied data aligns on a word boundary; data is simply copied as specified by the offset values.

Offsets in MLTE are always character offsets.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetDataEncoded

Copies the text in a specified range, and if necessary, translates the text to match your application's preferred encoding.

```
OSStatus TXNGetDataEncoded (
   TXNObject iTXNObject,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset,
   Handle *oDataHandle,
   TXNDataType iEncoding
);
```

Parameters

`iTXNObject`
The text object that contains the data you want to copy.

`iStartOffset`
The absolute offset from which data copying should begin. You can use the `TXNGetSelection` (page 57) function to get the absolute offsets of the current selection.

`iEndOffset`
The absolute offset at which data copying should end. You can use the `TXNGetSelection` (page 57) function to get the absolute offsets of the current selection.

`oDataHandle`
A pointer to a handle. On return, a handle to the requested data. `TXNGetDataEncoded` allocates the handle as necessary. Your application must dispose of the handle. If there is no text in the range specified, the returned handle is `NULL`, and the function returns `noErr`.

`encoding`
The type of data to be encoded. See Supported Data Types (page 152) for a full description of possible values. You should specify either the `kTXNTextData` or `kTXNUnicodeTextData` constant. If the `iEncoding` parameter specifies an encoding different from that used to store the text data internally, the Conversion Manager translates the data to the specified type (text or Unicode). If the `iEncoding` parameter is not recognized, the data is returned in the current encoding. On systems that do not use ATSUI version 1.1 or later, the current encoding is the Mac OS encoding. Otherwise, the current encoding is Unicode.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
Offsets in MLTE are always character offsets.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetEventTarget

Obtains the current event target for a TXNObject.

```
OSStatus TXNGetEventTarget (
   TXNObject iTXNObject,
   HIObjectRef *oEventTarget
);
```

Parameters

iTXNObject

The text object.

oEventTarget

A pointer to an `HIObjectRef` that, on return, points to the current event target for the TXNObject specified by `iTXNObject`.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

This function obtains the current event target for the TXNObject specified by `iTXNObject`. Use this function to obtain the target and then install your own handlers.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNGetHIRect

Obtains the values for the current view, destination, or text rectangle.

```
OSStatus TXNGetHIRect (
   TXNObject iTXNObject,
   TXNRectKey iTXNRectKey,
   HIRect *oRectangle
);
```

Parameters

iTXNObject

The text object for the current text area. You can either call the function `TXNCreateObject` to allocate a text object or you can call the function `HITextViewGetTXNObject` (page 23) to obtain the text object associated with an HITextView.

iTXNRectKey

A value that specifies the rectangle you want the function to obtain. See `Rectangle Keys` (page 146) for a list of the constants you can supply.

oRectangle

On output, points to the `HIRect` data structure that contains the coordinates for the requested rectangle.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetIndexedRunInfoFromRange

Gets information about a run in a range of data.

```
OSStatus TXNGetIndexedRunInfoFromRange (
   TXNObject iTXNObject,
   ItemCount iIndex,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset,
   TXNOffset *oRunStartOffset,
   TXNOffset *oRunEndOffset,
   TXNDataType *oRunDataType,
   ItemCount iTypeAttributeCount,
   TXNTypeAttributes *ioTypeAttributes
);
```

Parameters

`iTXNObject`
The text object for the current text area.

`iIndex`
The value that corresponds to the run for which you want to get information. You call the `TXNCountRunsInRange` (page 34) function to get the number of runs in a range. The `iIndex` parameter is zero-based, so its possible values are from 0 to the number of runs in a range minus 1. Note that the index is relative to the first run in the range specified by the `iStartOffset` and `iEndOffset` parameters, not for the entire document.

`iStartOffset`
The offset at which you want to start to obtain run information. This value must be the same value that you passed previously to the function `TXNCountRunsInRange`.

`iEndOffset`
The offset at which you want run information to end. This value must be the same value that you passed previously to the function `TXNCountRunsInRange`.

`oRunStartOffset`
On return, a pointer to a value that identifies the start of run relative to the beginning of the text, not the beginning of the range you specified in the `iStartOffset` parameter.

`oRunEndOffset`
On return, a pointer to a value that identifies the end of the run relative to the beginning of the text, not the beginning of the range you specified in the `iStartOffset` parameter.

`oRunDataType`
On return, a pointer to a value that identifies the type of data in the run. See Supported Data Types (page 152) for a description of possible values.

`iTypeAttributeCount`
The number of font attributes.

ioTypeAttributes

A pointer to a structure of type `TXNTypeAttributes`. On input, you specify the attribute (such as size) in the `tag` field and the attribute size in the `size` field. You can pass `NULL` for the `data` field only if the size of the returned value <= 4. Otherwise a pointer to an appropriately sized block of data must be placed in one of the other members of the `ioTypeAttributes` union, such as, `dataPtr`, `atsuFeatures`, `atsuVariations`. On return, the `data` field contains the attribute data. The `data` field is a union that serves either as a 32-bit integer or a 32-bit pointer, depending on the `size` field.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

You should first call the `TXNCountRunsInRange` function to get the count. The `TXNTypeAttributes` structure must specify the text attribute in which the application is interested. In other words, the `tag` and `size` fields must be set.

Offsets in MLTE are always character offsets.

If a tag specified in the `ioTypeAttributes` parameter is not recognized by MLTE—that is, the tag isn't a `TXNTag` value, a `TXNTypeRunAttributes` value, or a valid style run attribute tag (`ATSUAttributeTag`)—then the constant `kTXNAttributeTagInvalidForRunErr` is returned in the `ioTypeAttributes->data.dataValue` field for that particular tag. The function continues to process the remaining tags, but returns the result code `kTXNSomeOrAllTagsInvalidForRunErr`. Both of these values (`kTXNAttributeTagInvalidForRunErr` and `kTXNSomeOrAllTagsInvalidForRunErr`) are used either when a tag is not recognized or a tag is recognized but some error occurred in trying to obtain the attribute.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNGetLineCount

Gets the total number of lines in a text object.

```
OSStatus TXNGetLineCount (
   TXNObject iTXNObject,
   ItemCount *oLineTotal
);
```

Parameters

iTXNObject

The text object that identifies the document whose line count you want to get.

oLineTotal

On return, a pointer to the number of lines in the text object.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetLineMetrics

Gets information about line width and height for a specified line of data in a text object.

```
OSStatus TXNGetLineMetrics (
   TXNObject iTXNObject,
   unsigned long iLineNumber,
   Fixed *oLineWidth,
   Fixed *oLineHeight
);
```

Parameters

iTXNObject
: The text object that identifies the document whose line metrics you want to get.

iLineNumber
: A value that specifies the line whose metrics you want to retrieve. You should use the `TXNGetLineCount` (page 56) function to determine how many lines are in the text object so that you specify a valid line number. Line numbers start at 0.

oLineWidth
: On return, a pointer to the width of the line, in pixels.

oLineHeight
: On return, a pointer to the height of the line, in pixels.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
You can use height and width information to adjust the size of the text object's frame.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetSelection

Gets the absolute offsets of the current selection.

```
void TXNGetSelection (
   TXNObject iTXNObject,
   TXNOffset *oStartOffset,
   TXNOffset *oEndOffset
);
```

Parameters

iTXNObject
: The text object for the current text area.

oStartOffset
On return, a pointer to the absolute starting offset of the current selection.

oEndOffset
On return, a pointer to the absolute ending offset of current selection.

Discussion
Offsets in MLTE are always character offsets. Each embedded graphics or sound object is counted as one character.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTMetaData

Declared In
`MacTextEditor.h`

TXNGetSleepTicks

Reports the appropriate amount of time to allot to background processes, depending on the state of the window.

```
UInt32 TXNGetSleepTicks (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject
The text object that identifies the current document.

Return Value
The appropriate wait time, in ticks. You pass this value to the `WaitNextEvent` function.

Discussion
In a cooperative processing environment, your application must determine how much time to give to background processes. A tick is 1/60 of a second.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetSpellCheckAsYouType

Determines whether the “Spell Check as You Type” feature is enabled.

```
Boolean TXNGetSpellCheckAsYouType (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object to query.

Return Value

A Boolean whose value is `true` if the “Spell Check as You Type” feature is enabled; otherwise, `false`.

Discussion

Call `TXNSetSpellCheckAsYouType` (page 81) to enable or disable the “Spell Check as You Type” feature.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNGetTXNObjectControls

Gets the current formatting and privileges attributes (such as justification, line direction, tab values, and read-only status) for a text object.

```
OSStatus TXNGetTXNObjectControls (
   TXNObject iTXNObject,
   ItemCount iControlCount,
   const TXNControlTag iControlTags[],
   TXNControlData oControlData[]
);
```

Parameters

iTXNObject

The text object that identifies the document to be activated.

iControlCount

The number of items in the `iControlTags` array.

iControlTags

An array of values that specify the kind of formatting information you want returned in the `oControlData` array. See Formatting and Privileges Settings (page 129) for a description of possible values.

oControlData

An array of `TXNControlData` (page 98) unions. On return, the array contains the information that was requested through the `iControlTags` array. Your application must allocate the `oControlData` array.

Return Value

A result code. See “MLTE Result Codes” (page 160).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetViewRect

Gets the rectangle that describes the current view of the document.

```
void TXNGetViewRect (
   TXNObject iTXNObject,
   Rect *oViewRect
);
```

Parameters

`iTXNObject`
: The text object for the current text area.

`oViewRect`
: On return, a pointer to a rectangle that describes the view of the document. The area described by the `oViewRect` parameter does not include the area that encloses the scroll bars. The coordinates of this rectangle are local to the window.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetWindowRef

Returns a reference to the window to which the specified text object is attached.

```
WindowRef TXNGetWindowRef (
   TXNObject iTXNObject
);
```

Parameters

`iTXNObject`
: The text object for which you want to obtain a window reference. You can call the function `TXNCreateObject` to allocate a text object.

Return Value
The window to which the text object is attached. Returns `NULL` if no window is attached.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGrowWindow

Adjusts the size of a window in response to mouse-down events in the size region of the window.

```
void TXNGrowWindow (
   TXNObject iTXNObject,
   const EventRecord *iEvent
);
```

Parameters

iTXNObject
: The text object for the current text area.

iEvent
: A pointer to the event record that contains the mouse-down event you want to apply to the window.

Discussion

The text object must be contained in a window and not a subframe of the window; otherwise the function does not adjust the size of the window. Before you call `TXNGrowWindow`, you should make sure that the front window belongs to your application.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNHIPointToOffset

Gets the offset value that corresponds to a point in local coordinates.

```
OSStatus TXNHIPointToOffset (
   TXNObject iTXNObject,
   const HIPoint *iHIPoint,
   TXNOffset *oOffset
);
```

Parameters

iTXNObject
: The text object for which you want to obtain an offset value. You can either call the function `TXNCreateObject` to allocate a text object or you can call the function `HITextViewGetTXNObject` (page 23) to obtain the text object associated with an HITextView.

iHIPoint
: The local coordinates of the point for which you want to obtain the offset value.

oOffset
: On output, a pointer to the offset that corresponds to the value of the `iHIPoint` parameter.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

Offsets in MLTE are always character offsets.

Availability
Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNIdle

Does idle time processing, such as flashing the cursor.

```
void TXNIdle (
   TXNObject iTXNObject
);
```

Parameters

`iTXNObject`
: The text object that identifies the current document.

Discussion
Before you call the `TXNIdle` function, you should make sure that the front window belongs to your application.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNInitTextension

Initializes MLTE.

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

Parameters

`iDefaultFonts`
: An array of `TXNMacOSPreferredFontDescription` structures. You use this to specify a table of font information that includes the font family ID, point size, style, and script code. The table can be `NULL` or can have an entry for any script for which you would like to designate a default font. To designate that MLTE should use the default settings for a specified script, you need to supply a valid script code value in the `TextEncoding` field of the font description structure and a value of `kTXNUseScriptDefaultValue` in all other fields of the structure. In Mac OS X, the default settings are read from the Theme settings. In Mac OS 9, the default settings are read from the Script Manager.

`iCountDefaultFonts`
: The number of scripts for which you are designating a default font in the `iDefaultFonts` array.

`iUsageFlags`
: A value that specifies whether embedded objects should be supported. You can also specify whether MLTE should use QuickDraw as the imaging system and whether temporary memory should be used. See Initialization Option Masks (page 141) for a description of possible values.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
For systems prior to Mac OS X v10.3, you should call this function along with any other initialization calls you make when your application starts up. If you call this function a second time, it has no effect; the defaults you set up the first time you called the `TXNInitTextension` function are in effect until you call the `TXNTerminateTextension` (page 175) function.

For Mac OS X v10.3 and later, you do not have to call this function. However, you may want to call this function to set default fonts that are different from the system default font or to enable multimedia support.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNIsScrapPastable

Tests whether the Clipboard contains data that is supported by MLTE.

```
Boolean TXNIsScrapPastable (
   void
);
```

Return Value
Returns `true` if the data type on the Clipboard is supported.

Discussion
You can call the `TXNIsScrapPastable` function to determine if the Paste item in the Edit menu should be active.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNIsSelectionEmpty

Determines whether the current selection is empty.

```
Boolean TXNIsSelectionEmpty (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object for the current text area.

Return Value

A `Boolean` value. It is `true` if the current selection is empty.

Discussion

You can use the `TXNIsSelectionEmpty` function to determine whether your application should enable the Cut, Copy, and Clear items in the Edit menu.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNKeyDown

Processes a key-down event.

```
void TXNKeyDown (
   TXNObject iTXNObject,
   const EventRecord *iEvent
);
```

Parameters

iTXNObject

The text object that identifies the active document.

iEvent

A pointer to the event record that contains the key-down event you want handled. You cannot pass `NULL`.

Discussion

Before you call the `TXNKeyDown` function, you should make sure that the front window belongs to your application. Text input occurs in the text object's window. This is always the case unless the application has requested text input through a bottom-line window (a small window that appears at the bottom of the screen) or has turned off the Text Services Manager (TSM).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNOffsetToHIPoint

Obtains the local coordinates of the point that corresponds to a specified offset of a text object.

```
OSStatus TXNOffsetToHIPoint (
   TXNObject iTXNObject,
   TXNOffset iOffset,
   HIPoint *oHIPoint
);
```

Parameters

iTXNObject

The text object for which you want to obtain the local coordinates of a point. You can either call the function `TXNCreateObject` to allocate a text object or you can call the function `HITextViewGetTXNObject` (page 23) to obtain the text object associated with an HITextView.

iOffset

The offset value of the point for which you want to obtain the local coordinates.

oPoint

On output, a pointer to the local coordinates of the point that corresponds to the value of the `iOffset` parameter.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

Offsets in MLTE are always character offsets.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNPageSetup

Displays the Page Setup dialog for the current default printer and manages changes, such as reformatting the text, in response to page layout changes.

```
OSStatus TXNPageSetup (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object for the active document.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNPaste

Pastes the contents of the private MLTE scrap into the text object.

```
OSStatus TXNPaste (
   TXNObject iTXNObject
);
```

Parameters

`iTXNObject`
The text object that identifies the current document.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
Before you call `TXNPaste`, you can call the `TXNIsScrapPastable` (page 63) function to determine if the current scrap contains data supported by MLTE.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNPrint

Prints the document so it is formatted to fit the page size selected for the printer.

```
OSStatus TXNPrint (
   TXNObject iTXNObject
);
```

Parameters

`iTXNObject`
The text object that identifies the document you want to print.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNReadFromCFURL

Reads data from a `CFURLRef` into a TXNObject.

```
OSStatus TXNReadFromCFURL (
   TXNObject iTXNObject,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset,
   CFDictionaryRef iDataOptions,
   CFURLRef iFileURL,
   CFDictionaryRef *oDocumentAttributes
);
```

Parameters

iTXNObject

The text object into which data read from `iFileURL` is to be added.

iStartOffset

The offset in `iTXNObject` at which to start placing data read from `iFileURL`.

iEndOffset

The offset in `iTXNObject` at which to stop placing data read from `iFileURL`.

iDataOptions

Options for reading the data. See Data Option Key Value Constants (page 120) for a list of the supported options. If this parameter is `NULL`, the data is read in using MLTE's native format.

iFileURL

A `CFURLRef` to the data that is to be added to the text object specified by `inTXNObject`.

oDocumentAttributes

A value of type CFDictionary that, on return, contains the document attributes present in the data stream, if the file format supports them; otherwise this parameter is `NULL`. The native MLTE file format and RTF support embedded document attributes. See Document Attribute Keys (page 122) for a list of supported attributes. If this parameter is `NULL`, no document attributes are written out.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

This function reads data from a file or a special file bundle (directory) into a text object. Offset parameters are used to specify whether the new data is inserted, appended or replaces an existing data range in the text object. Clients can specify the document format and encoding of the data using the `iDataOptions` parameter. This functions also returns the document attributes present in the data stream. Document attributes are supported only for the rich text file formats supported by MLTE, which are RTF and MLTE native file format.

If the caller passes a pointer to a `CFDictionaryRef`, this function returns a reference to a dictionary of attributes, if there is one, that the caller is responsible for releasing. In all other cases, this function sets the reference to `NULL`. Here is some sample code:

```
CFDictionaryRef oDocumentAttributes = NULL;
status = TXNReadFromCFURL (....,&oDocumentAttributes);
if (oDocumentAttributes != NULL) ::CFRelease(oDocumentAttributes);
```

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNRecalcTextLayout

Recalculates the text layout based on new view and destination rectangles.

```
void TXNRecalcTextLayout (
   TXNObject iTXNObject
);
```

Parameters

`iTXNObject`
The text object whose layout you want to recalculate.

Discussion
You can call the function `TXNRecalcTextLayout` if you call the function `TXNSetRectBounds` (page 174) with the `iUpdate` parameter set to `false`. `TXNRecalcTextLayout` recalculates the text layout as well as where the scroll bars, if any, should be placed. When you call `TXNRecalcTextLayout`, MLTE generates an update event to redraw the text object.

Availability
Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNRedo

Redoes the last command.

```
void TXNRedo (
   TXNObject iTXNObject
);
```

Parameters

`iTXNObject`
The text object for the document you want to examine.

Discussion
If the user undoes an action and then undoes it again, the second undo is the same as a redo.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNRegisterScrollInfoProc

Installs or uninstalls a scrolling callback function on a text object.

```
void TXNRegisterScrollInfoProc (
   TXNObject iTXNObject,
   TXNScrollInfoUPP iTXNScrollInfoUPP,
   SRefCon iRefCon
);
```

Parameters

iTXNObject
: The text object on which you want to install a callback to scroll text.

iTXNScrollInfoUPP
: A universal procedure pointer to the callback function you want MLTE to call whenever a scroll bar for the text object must be updated.

iRefCon
: A 32-bit value that is passed to your callback.

Discussion

You can call the function `TXNRegisterScrollInfoProc` to install a text-scrolling callback on a text object. This is useful if your application draws and handles its own scrolling widgets. Once you register your callback (`TXNScrollInfoUPP`), MLTE invokes your callback each time it is necessary to update the values and maximum values of your scrolling widget. For example when the user presses the Return key to add a new line, MLTE calculates a new maximum value for the text object. Your callback is then called with the newly-calculated maximum value. To turn off your callback call the function `TXNRegisterScrollInfoProc` with a value of `NULL` for the `iTXNScrollInfoUPP` parameter.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNResizeFrame

Resizes the view and destination rectangles.

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

Parameters

iTXNObject
: The text object for the current text area.

iWidth
: The new width of the view and destination rectangles in pixels.

iHeight
: The new height of the view and destination rectangles in pixels.

iTXNFrameID
: The frame ID of the frame associated with the view and destination rectangles you want to resize. You obtain the frame ID when you call the `TXNNewObject` (page 170) function.

Discussion
You need to call the function `TXNSetFrameBounds` (page 77) if you want to reset the frame bounds.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNRevert

Reverts to the last saved version of a document.

```
OSStatus TXNRevert (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject
The text object for the active document.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
You can use the `TXNRevert` function with files that contain only text as well as files that were created using MLTE. If the file was not previously saved, the document reverts to an empty document. To revert to data that is embedded in a private file type, use the `TXNSetSelection` (page 80) function to select all of the current data and then use the `TXNSetDataFromFile` (page 173) function to read in the old data.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNScroll

Scrolls the text within a view rectangle of the specified text object.

```
OSStatus TXNScroll (
   TXNObject iTXNObject,
   TXNScrollUnit iVerticalScrollUnit,
   TXNScrollUnit iHorizontalScrollUnit,
   long *ioVerticalDelta,
   long *ioHorizontalDelta
);
```

Parameters

iTXNObject
The text object whose text you want to scroll.

iVerticalScrollUnit

Specifies the units to use for the `ioVerticalDelta` parameter. Pass `kTXNScrollUnitsInPixels` to specify pixels, pass `kTXNScrollUnitsInLines` to specify a count of lines, and pass `kTXNScrollUnitsInViewRects` to specify the height of the current view rectangle (`viewRect`). Note that scrolling in line units is the slowest because each line must be measured by MLTE before the text scrolls. See Scroll Units (page 148) for more information.

iHorizontalScrollUnit

Specifies the units to use for the `ioHorizontalDelta` parameter. Pass `kTXNScrollUnitsInPixels` to specify pixels, pass `kTXNScrollUnitsInLines` to specify a count of lines, and pass `kTXNScrollUnitsInViewRects` to specify the height of the current view rectangle (`viewRect`). Note that scrolling in line units is the slowest because each line must be measured by MLTE before the text scrolls. See Scroll Units (page 148) for more information.

ioVerticalDelta

On input, the number of units by which to scroll in the vertical direction. You specify the units in the `iVerticalScollUnit` parameter. On output, the number of units actually scrolled in the vertical direction. A positive value indicates to move the text in a downward direction.

ioHorizontalDelta

On input, the number of units by which to scroll in the horizontal direction. You specify the units in the `iHorizontalScrollUnit` parameter. On output, the number of units actually scrolled in the horizontal direction. A positive value indicates to move the text to the right.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

The function `TXNScroll` moves the text within the view rectangle of the specified text object by the designated number of units. You can use this function to scroll the text in a text object in response to user input in a control other than the standard scroll bars that MLTE supplies.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNSelectAll

Selects all data in the frame of a text object.

```
void TXNSelectAll (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object for the current text area.

Discussion

You can check whether your application should enable the Select All menu item by calling the `TXNDataSize` (page 37) function to check if the text object contains any data.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetActionNameMapper

Sets a callback that MLTE uses to obtain the localized string representing an action or an action group.

```
OSStatus TXNSetActionNameMapper (
   TXNObject iTXNObject,
   TXNActionNameMapperUPP iStringForKeyProc,
   const void *iUserData
);
```

Parameters

`iTXNObject`
The text object for which the callback is to be set.

`iStringForKeyProc`
The callback.

`iUserData`
A pointer to user-defined data that will help you map the action key to a string.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
If you have asked MLTE to handle updating for the Redo and Undo commands in the Edit menu, you should call this function so that MLTE can call your callback, which provides the correct string for each of those commands.

When MLTE's handler for `kEventClassCommand`/`kEventCommandUpdateStatus` is called for the Redo or Undo command, MLTE checks to see if a `TXNActionNameMapperProc` has been installed. If a `TXNActionNameMapperProc` is installed, it is called to get the correct string for updating the menu item. The client can used the action name and the command ID to determine the appropriate string.

For information on a `TXNActionNameMapperProc`, see TXNActionNameMapperProcPtr (page 89).

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetBackground

Sets the background on which the text object's data is drawn.

```
OSStatus TXNSetBackground (
   TXNObject iTXNObject,
   const TXNBackground *iBackgroundInfo
);
```

Parameters

iTXNObject
The text object that identifies the document to be activated.

iBackgroundInfo
A pointer to a structure that describes the background.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Version Notes
MLTE supports only color as the background.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetCommandEventSupport

Enables and disables support for menu commands in MLTE.

```
OSStatus TXNSetCommandEventSupport (
   TXNObject iTXNObject,
   TXNCommandEventSupportOptions iOptions
);
```

Parameters

iTXNObject
The TXNObject.

iOptions
The menu commands for which support is to be enabled or disabled. For possible values, see Command Event Support Options (page 116).

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
When `TXNAttachObjectToWindow` (page 165) or `TXNSetEventTarget` (page 75) is called to associate an MLTE object with an HIObject that can serve as an event target, most handlers are installed and activated immediately.

However, when the handlers for the kEventClassCommand class are installed, they are not activated. Call this function to activate the handlers for this class, which provide support for the menu commands. This approach means that an application can install handlers on top of these and be sure that enabling or disabling the MLTE handlers does not change the order of the handler chain.

Availability
Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetContextualMenuSetup

Provides a callback function that is called before MLTE displays its contextual menu.

```
OSStatus TXNSetContextualMenuSetup (
   TXNObject iTXNObject,
   TXNContextualMenuSetupUPP iMenuSetupProc,
   const void *iUserData
);
```

Parameters

`iTXNObject`
: The text object.

`iMenuSetupProc`
: The callback. For more information, see `NewTXNContextualMenuSetupUPP` (page 27).

`iUserData`
: A pointer to user-defined data that will be passed to the callback specified by the `iMenuSetupProc` parameter.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
The callback function specified by `iMenuSetupProc` is called just before MLTE displays its contextual menu. The menu that is passed to the callback contains MLTE-specific items only. The client items and handlers should be installed each time the callback is called.

When the callback is called, MLTE has selected the word the user clicked with the Option key pressed. For convenience, the TXNObject associated with the callback is passed to the callback as well as the data specified by `iUserData`.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetData

Replaces a range of data (text, graphics, and so forth).

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

Parameters

`iTXNObject`
: The text object that identifies the document in which you want to replace data.

`iDataType`
: The type of the replacement data. See Supported Data Types (page 152) for a description of possible values.

`iDataPtr`
: A pointer to the data that will replace the data that is in the range specified by the `iStartOffset` and `iEndOffset` parameters.

`iDataSize`
: The size of the data (in bytes) to which `iDataPtr` points.

`iStartOffset`
: The beginning of the range of data to replace. You can use the `TXNGetSelection` (page 57) function to get the absolute offsets of the current selection.

`iEndOffset`
: The end of the range to replace. You can use the `TXNGetSelection` (page 57) function to get the absolute offsets of the current selection. If you want to insert text, the ending and starting offsets should be the same value.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
If you have a text object that has word wrap disabled, and you want to avoid horizontal scrolling, you can try the following. After you call the function `TXNSetData`, call `TXNSetSelection` (page 80) with the value of the ending offset set to what it was before you called `TXNSetData`. Then, call the function `TXNShowSelection` (page 83) to scroll the text back into view.

Offsets in MLTE are always character offsets.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code
QTMetaData

Declared In
`MacTextEditor.h`

TXNSetEventTarget

Sets a Carbon Event target for MLTE Carbon Event handlers.

```
OSStatus TXNSetEventTarget (
   TXNObject iTXNObject,
   HIObjectRef iEventTarget
);
```

Parameters

`iTXNObject`
: The TXNObject.

`iEventTarget`
: The `HIObjectRef` that is to be set as the event target for all of the Carbon Event handlers of the TXNObject specified by `iTXNObject`.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
The default target for a TXNObject that is not contained in an HITextView is the window of the TXNObject. Call this function when you want to override or set the Carbon Event target for a TXNObject.

Note that if the TXNObject already has a default target when this function is called, the handlers are removed from the old target before the new handlers are installed.

When this function returns, these handlers for Carbon Events are installed and active for the `kEventClassTextInput` class:

- `kEventTextInputUpdateActiveInputArea`
- `kEventTextInputUnicodeForKeyEvent`
- `kEventTextInputUnicodeText`
- `kEventTextInputOffsetToPos`
- `kEventTextInputPosToOffset`
- `kEventTextInputGetSelectedText`

When this function returns, these handlers for Carbon Events are installed and active for the `kEventClassTSMDocumentAccess` class:

- `kEventTSMDocumentAccessGetLength`
- `kEventTSMDocumentAccessGetSelectedRange`
- `kEventTSMDocumentAccessGetCharactersPtr`
- `kEventTSMDocumentAccessGetCharactersPtrForLargestBuffer`
- `kEventTSMDocumentAccessGetCharacters`
- `kEventTSMDocumentAccessGetFont`
- `kEventTSMDocumentAccessGetGlyphInfo`

When this function returns, these handlers for Carbon Events are installed and active for the `kEventClassFont` class:

- `kEventFontPanelClosed`
- `kEventFontSelection`

When this function returns, these handlers for Carbon Events are installed and inactive by default for the `kEventClassCommand` class:

- `kEventProcessCommand`
- `kEventCommandUpdateStatus`

The `kEventClassCommand` handlers support the following commands:

- `kHICommandUndo`
- `kHICommandRedo`
- `kHICommandSelectAll`
- `kHICommandCut`
- `kHICommandCopy`
- `kHICommandPaste`
- `kHICommandClear`
- `kHICommandShowSpellingPanel`
- `kHICommandCheckSpelling`
- `kHICommandChangeSpelling`
- `kHICommandCheckSpellingAsYouType`
- `kHICommandIgnoreSpelling`
- `kHICommandLearnWord`

Activate command support by calling `TXNSetCommandEventSupport` (page 73) with the appropriate options.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetFrameBounds

Changes the boundaries of a text object's frame.

```
void TXNSetFrameBounds (
   TXNObject iTXNObject,
   SInt32 iTop,
   SInt32 iLeft,
   SInt32 iBottom,
   SInt32 iRight,
   TXNFrameID iTXNFrameID
);
```

Parameters

iTXNObject
: The text object for the current text area.

iTop
: The top boundary of the rectangle.

iLeft
: The left boundary of the rectangle.

iBottom
: The bottom boundary of the rectangle.

iRight
: The right boundary of the rectangle.

iTXNFrameID
: The frame ID of the frame you want to move. You obtain a frame ID when you call the `TXNNewObject` (page 170) function.

Discussion

If you want to change the view and destination rectangles, you should call the `TXNResizeFrame` (page 69) function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNSetHIRectBounds

Sets the view rectangle and/or the destination rectangle.

```
void TXNSetHIRectBounds (
   TXNObject iTXNObject,
   const HIRect *iViewRect,
   const HIRect *iDestinationRect,
   Boolean iUpdate
);
```

Parameters

iTXNObject
: The text object for the current text area. You can call the function `TXNCreateObject` to allocate a text object.

iViewRect
: A pointer to rectangle that contains the new coordinates for the view rectangle. If you do not want to change the view rectangle pass `NULL`. You cannot set the view rectangle for text objects into an HITextView, you can only set the destination rectangle.

iDestinationRect
: A pointer to a rectangle that contains the new coordinates for the destination rectangle. If you do not want to change the destination rectangle pass `NULL`.

iUpdate
: A value that specifies whether you want the text and scroll bars recalculated and redrawn. Pass `true` to recalculate and redraw; otherwise pass `false`. You must pass `false` for text objects into an HITextView.

Discussion

The view rectangle controls the text you see. The destination rectangle controls how text is laid out. You can specify coordinates for one or both rectangles. Scroll bars are drawn inside the view rectangle.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNSetScrollbarState

Sets the state of the scroll bars so they are drawn correctly in response to activate events.

```
OSStatus TXNSetScrollbarState (
   TXNObject iTXNObject,
   TXNScrollBarState iActiveState
);
```

Parameters

iTXNObject
: The text object that identifies the document you want activated. You can call the function `TXNCreateObject` to allocate a text object.

iActiveState
: A value that indicates the state of the scroll bars. See Scroll Bar States (page 147) for a description of possible values. If you pass the `kScrollBarsAlwaysActive` constant, the scroll bars are always active, whether or not the frame text area currently has keyboard focus. Passing `kScrollBarsAlwaysActive` can be useful for a window such as a dialog that may contain multiple text areas, each of which may have a scrollable frame. If you pass `kScrollBarsSyncWithFocus`, MLTE synchronizes the activity state of the scroll bars with the focus state of the frame. Therefore, only when the frame has keyboard focus does it have active scroll bars. A value of `kScrollBarsSyncWithFocus` is the default and is typically recommended if you have only one frame per window.

Return Value

A result code. See "MLTE Result Codes" (page 160). This function returns an error if the text object is in an HITextView.

Discussion
This function is a macro that calls the function `TXNActivate` with the `TXNFrameID` parameter set to `0`. You typically call `TXNSetScrollbarState` in response to an activate event. If the text object was previously inactive, `TXNSetScrollbarState` removes any visual indication of its prior inactive state (such as a dimmed or framed selection area or inactive scroll bars). Before you call the `TXNSetScrollbarState` function, you should make sure that the window belongs to your application.

The `TXNSetScrollbarState` function does not change the keyboard focus. This means your application can have a text area that is not focused, but in which the scroll bars are active. This lets application users scroll the inactive text without changing the focus from another text area.

If you want to display a text area that has both keyboard focus and active scroll bars, you must call the `TXNFocus` (page 47) function immediately before you call the `TXNSetScrollbarState` function. Note that MLTE does not retain information about keyboard focus. So if, for example, you set the keyboard focus on a text area and the window containing the text area becomes deactivated, you must call the `TXNFocus` (page 47) function when the window becomes activated again.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetSelection

Specifies the selection range or the position of the insertion point.

```
OSStatus TXNSetSelection (
   TXNObject iTXNObject,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset
);
```

Parameters

`iTXNObject`
The text object that identifies the document for which you want to set the selection range or insertion point position.

`iStartOffset`
The new starting offset. Offset values are character offsets.

`iEndOffset`
The new ending offset. Offset values are character offsets.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
You can use the `TXNSetSelection` function to highlight an initial default value in a document, such as a data-entry form, or to position the insertion point at the start of the field where you want the user to enter a value. To position the insertion point, specify the same value for the `iStartOffset` and `iEndOffset` parameters.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetSpellCheckAsYouType

Enables and disables the “Spell Check as You Type” feature.

```
OSStatus TXNSetSpellCheckAsYouType (
   TXNObject iTXNObject,
   Boolean iActivate
);
```

Parameters

`iTXNObject`
The text object for which the “Spell Check as You Type” feature is to be enabled or disabled.

`iActivate`
A Boolean whose value is `true` to enable the “Spell Check as You Type” feature or `false` to disable it.

Return Value
A result code. See “MLTE Result Codes” (page 160). If `TXNSetCommandEventSupport` (page 73) has not been called to enable the event handler for spell checking, an error is returned.

Availability
Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetTXNObjectControls

Sets formatting and privileges attributes (such as justification, line direction, tab values, and read-only status) that apply to the entire text object.

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

Parameters

`iTXNObject`
The text object that identifies the document for which you want to set formatting and privileges attributes.

`iClearAll`
A `Boolean` value. If you set this to `true`, all formatting and privileges attributes are reset to their default value. That is, `true` clears existing tags and resets each to its default value.

iControlCount
: The number of items in the `iControlTags` array.

iControlTags
: An array of values that specifies kind of data that is passed in the `iControlData` parameter. See Formatting and Privileges Settings (page 129) for a description of possible values. On systems that use Apple Type Services for Unicode Imaging (ATSUI), you can also pass any of the following ATSUI attribute tag constants:

- `kATSULineDirectionTag`
- `kATSULineJustificationFactorTag`
- `kATSULineFlushFactorTag`
- `kATSULineBaselineValuesTag`
- `kATSULineLayoutOptionsTag`
- `kATSUCGContextTag`

See the ATSUI documentation for a description of these ATSUI constants.

iControlData
: An array of `TXNControlData` (page 98) unions that contain the information your application wants to set. The value you supply to the `iControlTags` parameter specifies how the union of type `TXNControlData` is treated. You must make sure that the value you assign to the `iControlData` parameter is the appropriate type implied by the value you passed in the `iControlTags` parameter.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
On systems that use Apple Type Services for Unicode Imaging (ATSUI), the ATSUI line control attribute tags can be passed to this function in the `iControlTag` parameter. This is the case for all the ATSUI tags except `kATSULineRotationTag`. ATSUI tags are applied to the entire text object.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetTypeAttributes

Sets text attributes (such as size and style) for the current selection or the text defined by a range you specify.

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

Parameters

iTXNObject
: The text object that contains the current selection.

iAttrCount

The number of font attributes in the `iAttributes` array.

iAttributes

An array of `TXNTypeAttributes` structures in which you specify the attributes you want to set. Values passed in the `iAttributes` array that are less than or equal to `sizeof(UInt32)` are passed by value. Values greater than `sizeof(UInt32)` are passed as a pointer. That is, the third field of the `TXNTypeAttributes` structure is a union that serves as either a 32-bit integer or a 32-bit pointer.

iStartOffset

The starting offset at which you want the application to begin setting attributes. If you want to use the current selection, set `iStartOffset` to `kTXNUseCurrentSelection`.

iEndOffset

The offset at which you want the application to stop setting attributes. If you want to use the current selection, set `iEndOffset` to `kTXNUseCurrentSelection`.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

You can use this function to clear ATSUI font features and ATSUI font variations. To clear either the features or the variations, you must `OR` the `kTXNClearTheseFontFeatures` constant with the current ATSUI font feature or font variation setting. See Clearance Settings (page 116) for more information.

Offsets in MLTE are always character offsets.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNShowSelection

Scrolls the current selection into view.

```
void TXNShowSelection (
   TXNObject iTXNObject,
   Boolean iShowEnd
);
```

Parameters

iTXNObject

The text object for the current text area.

iShowEnd

A `Boolean` value. If you set this to `true`, the end of the selection is scrolled into view. Otherwise, the beginning of the selection is scrolled into view.

Discussion

You can use this to scroll text into view after you have called the `TXNFind` (page 45) function and the matching text is not in the current view of the text object.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNTSMCheck

Checks to see if the Text Services Manager (TSM) is active.

Not Supported

```
Boolean TXNTSMCheck (
    TXNObject iTXNObject,
    EventRecord *iEvent
);
```

Parameters

iTXNObject
: The text object that identifies the current document. Pass `NULL` when there is no active text object and you want to check for TSM activity.

iEvent
: A pointer to an event record structure. This can be `NULL`.

Return Value
A `Boolean` value. It is `true` if TSM is active and `false` if TSM is not active.

Discussion
The purpose of this function is to ensure input methods have enough time to respond. Call this when the `WaitNextEvent` function returns `false` or there is no active text object. For a Unicode application, if the event is a key-down event, `TXNTSMCheck` will also process the event and set the event to `NULL`.

Declared In
`MacTextEditor.h`

TXNUndo

Undoes the last command.

```
void TXNUndo (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject
: The text object for the document you want to examine.

Discussion
The undo stack is 32 levels deep. That is, undoable actions are tracked until the total count is 32. If a user undoes two actions, the Redo command must be used twice to get back to the original state. If more than 32 actions are performed, the oldest actions are forgotten as each new action takes place.

If the user performs a new action after choosing Redo from the Edit menu, the redone action is no longer available to be undone. For example, a user performs the following actions: types some text, cuts some text, pastes some text, types some text; undoes the last typing action, and undoes the paste operation; redoes the paste; types some new text. After the new text has been typed, the undo stack contains the first text that was typed, the cut action, and the new text that was just typed.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNUpdate

Redraws everything in a frame in response to an update event.

```
void TXNUpdate (
   TXNObject iTXNObject
);
```

Parameters

`iTXNObject`
    The text object that identifies the document to be updated.

Discussion
This function calls the Window Manager `BeginUpdate` and `EndUpdate` functions for the window that you pass to the `TXNNewObject` function. You shouldn't use it for windows that contain something else besides the text object. If the window contains something in addition to the text object, you should use the `TXNDraw` (page 167) function instead of the `TXNUpdate` function.

Before you call `TXNUpdate`, you should make sure that the window belongs to your application. The `TXNUpdate` function redraws any content that needs updating regardless of the layer in which your window is located.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNVersionInformation

Gets the version number of MLTE and the set of features in this version.

```
TXNVersionValue TXNVersionInformation (
   TXNFeatureBits *oFeatureFlags
);
```

Parameters

oFeatureFlags

On return, a pointer to a value that indicates the set of features in use in this version. See Frame Option Bits (page 133) and ATSUI Feature Masks (page 113) for a description of possible values.

Return Value

A value that specifies the version of MLTE. See the description of the `TXNVersionValue` data type.

Discussion

If the bit `kTXNWillDefaultToATSUIBit` is set, then by default MLTE uses ATSUI to image and measure text and uses Unicode to store characters. If the bit `kTXNWillDefaultToCarbonEventBit` is set, then, by default, MLTE uses Carbon events and Apple events are not supported.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNWriteRangeToCFURL

Writes a range of a text object to a file or to a special file bundle.

```
OSStatus TXNWriteRangeToCFURL (
   TXNObject iTXNObject,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset,
   CFDictionaryRef iDataOptions,
   CFDictionaryRef iDocumentAttributes,
   CFURLRef iFileURL
);
```

Parameters

iTXNObject

The text object having a range that is to be written.

iStartOffset

The offset in `iTXNObject` at which to start writing data to `iFileURL`.

iEndOffset

The offset in `iTXNObject` at which to stop writing data to `iFileURL`.

iDataOptions

A CFDictionaryRef that specifies options for writing out the data. See Data Option Key Value Constants (page 120) for a list of the supported options. If this parameter is `NULL`, the data is written out using MLTE's native format.

iDocumentAttributes

The document attributes that are to be embedded in the data stream. This parameter is only supported when writing out the data using one of the following formats: RTF and MLTE native format. Only the key / values defined in Document Attribute Keys (page 122) are written out. The content of the dictionary is ignored for any other format. If the dictionary is `NULL`, no attributes are added to the data stream.

iFileURL

`CFURLRef` for an existing file or directory, whichever is appropriate for the file type. On return, `iFileURL` contains a copy of the data in the given range for the `iTXNObject` with the format and encoding specified by `iDataOptions`.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

This function writes a range of a text object to a file or a special file bundle (directory). It supports several document formats and encodings, which can be specified in the data options dictionary. Clients can specify additional document attributes when data is written out using a file format, such as RTF and native MLTE file format) that supports such attributes.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNZoomWindow

Increases the size of the data displayed in a window in response to a click in the zoom box.

```
void TXNZoomWindow (
   TXNObject iTXNObject,
   SInt16 iPart
);
```

Parameters

iTXNObject

The text object for the current text area.

iPart

The location, in global coordinates, of the cursor at the time the user pressed the mouse button. You obtain this value from the Window Manager function `FindWindow`.

Discussion

Before you call the `TXNZoomWindow` function, you should make sure that the window belongs to your application. You should use the `TXNZoomWindow` function only when a text object has a viewable area that occupies the entire window; for example, if you passed `NULL` for the `iFrame` parameter when you called the `TXNNewObject` (page 170) function to create the text object. You cannot use `TXNZoomWindow` if the text object is contained in a subframe of a window.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

Callbacks

TXNActionKeyMapperProcPtr

Defines a pointer to an action key mapping function that customizes the Redo and Undo menu items with the specific action that can be redone or undone. (**Deprecated.** Use `TXNActionNameMapperProcPtr` (page 89) instead.)

```
typedef CFStringRef(* TXNActionKeyMapperProcPtr)
(
    TXNActionKey actionKey,
    UInt32 commandID
);
```

You would declare your function like this if you were to name it `MyTXNActionKeyMapperFunction`:

```
CFStringRef MyTXNActionKeyMapperCallback
(
    TXNActionKey actionKey,
    UInt32 commandID
);
```

Parameters

`actionKey`
A value of type `TXNActionKey` that indicates an editing action taken by the user.

`commandID`
The command ID of menu command chosen by the user.

Return Value
The localized string you want to map to the `TXNActionKey`.

Discussion
Your callback is invoked each time you ask MLTE to handle command updates for the Edit menu. MLTE calls your callback whenever it receives a Carbon event of class `kEventClassCommand` and event kind `kEventComandUpdateStatus` for the command ID `kHICommandUndo`. In other words, whenever the undo-command item in the menu needs to be updated. Your callback should examine the `actionKey` parameter and return an appropriately localized `CFStringRef` that describes the undo action. MLTE calls the function `SetMenuItemWithCFString` to update the menu item's text. You are responsible for releasing the `CFString`; MLTE does not call the function `CFRelease` on the string.

You provide a pointer to your action key mapping callback function when you build a dictionary to support Carbon events in MLTE. The pointer should be the value associated with the dictionary key `kTXNActionKeyMapperKey`. You then assign the dictionary to a `TXNCarbonEventInfo` (page 96) data structure. You treat the data structure as an object control. That is, you associate the `TXNCarbonEventInfo` data structure with a text object by calling the function `TXNSetTXNObjectControls` (page 81), supplying `kTXNUseCarbonEvents` in the `iControlsTags` parameter and the `TXNCarbonEventInfo` data structure in the `iControlData` parameter.

To provide a pointer to your action key mapping callback function, you use the `NewTXNActionKeyMapperUPP` function to create a universal procedure pointer (UPP) of type `TXNActionKeyMapperUPP`. You can do so with code similar to the following:

```
TXNActionKeyMapperUPP MyTXNActionKeyMapperUPP;
MyTXNActionKeyMapperUPP = NewTXNActionKeyMapperUPP
                        (&MyActionKeyMapperFunction);
```

When you are finished with your action key mapper callback function, you should use the `DisposeTXNActionKeyMapperUPP` function to dispose of the UPP associated with it. However, if you plan to use the same action key mapper callback later in your application, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNActionNameMapperProcPtr

Defines a pointer to an action name mapping function that customizes the Redo and Undo menu items with the specific action that can be redone or undone.

```
typedef CFStringRef TXNActionNameMapperProcPtr (
      CFStringRef actionName,
      UInt32 commandID,
      void * inUserData
);
```

If you name your function `MyTXNActionNameMapperProc`, you would declare it like this:

```
CFStringRef MyTXNActionNameMapperProc (
      CFStringRef actionName,
      UInt32 commandID,
      void * inUserData
);
```

Parameters

actionName
The name of the action.

commandID
The command ID of the menu command.

inUserData
User-defined data that was provided in the inUserData parameter when `InvokeTXNActionNameMapperUPP` (page 24) was called.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`MacTextEditor.h`

TXNContextualMenuSetupProcPtr

Defines a pointer to a contextual menu setup function.

```
typedef void TXNContextualMenuSetupProcPtr (
      MenuRef iContextualMenu,
      TXNObject object,
      void * inUserData
);
```

If you name your function `MyTXNContextualMenuSetupProc`, you would declare it like this:

```
void MyTXNContextualMenuSetupProc (
      MenuRef iContextualMenu,
      TXNObject object,
      void * inUserData
);
```

Parameters

iContextualMenu
: The MLTE contextual menu.

TXNObject
: The TXNObject for which `MyTXNContextualMenuSetupProc` was called.

inUserData
: User-defined data that was passed to `TXNSetContextualMenuSetup` (page 74).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`MacTextEditor.h`

TXNFindProcPtr

Defines a pointer to a find function that customizes a search tailored to your application's needs.

```
typedef OSStatus (*TXNFindProcPtr) (
    const TXNMatchTextRecord * matchData,
    TXNDataType iDataType,
    TXNMatchOptions iMatchOptions,
    const void * iSearchTextPtr,
    TextEncoding encoding,
    TXNOffset absStartOffset,
    ByteCount searchTextLength,
    TXNOffset * oStartMatch,
    TXNOffset * oEndMatch,
    Boolean * ofound,
    UInt32 refCon
);
```

If you name your function `MyTXNFindProc`, you would declare it like this:

```
OSStatus TXNFindProcPtr (
    const TXNMatchTextRecord * matchData,
    TXNDataType iDataType,
    TXNMatchOptions iMatchOptions,
    const void * iSearchTextPtr,
    TextEncoding encoding,
    TXNOffset absStartOffset,
    ByteCount searchTextLength,
    TXNOffset * oStartMatch,
    TXNOffset * oEndMatch,
    Boolean * ofound,
    UInt32 refCon
);
```

Parameters

matchData

A pointer to a `TXNMatchTextRecord` structure containing the text to match, the length of that text, and the text's encoding. Pass `NULL` if you are looking for a graphics, sound, or movie object.

iDataType

The type of data for which you want to search. See Supported Data Types (page 152) for a description of possible values.

iMatchOptions

A value that specifies the matching rules to use in the find operation. See Search Criteria Masks (page 149) for a description of possible values.

iSearchTextPtr

A pointer to the text to search.

encoding

The encoding of the text to search.

absStartOffset

The offset at which the search should begin. The constant `kTXNStartOffset` specifies the start of the object's data.

searchTextLength

The length, in bytes, of the text to search.

oStartMatch

On return, a pointer to the absolute offset that identifies the start of the match. Your function should set this to `kTXNUseCurrentSelection` if there is no match.

oEndMatch

On return, a pointer to the absolute offset that identifies the end of the match. Your function should set this to `kTXNUseCurrentSelection` if there is no match.

ofound

On return, a pointer to a `Boolean` value; `true` if a match is found.

refCon

An unsigned 32-bit integer your application can use as needed.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

You pass a pointer to your find callback function as a parameter to the `TXNFind` (page 45) function. To provide a pointer to your find callback function, you use the `NewTXNFindUPP` (page 27) function to create a universal procedure pointer (UPP) of type `TXNFindUPP`. You can do so with code similar to the following:

```
 TXNFindUPP MyTXNFindUPP;
```

```
MyTXNFindUPP = NewTXNFindUPP (&MyFindCallback)
```

When you are finished with your find callback function, you should use the `DisposeTXNFindUPP` (page 20) function to dispose of the UPP associated with it. However, if you plan to use the same find callback function in subsequent calls to the `TXNFind` function, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNScrollInfoProcPtr

Defines a pointer to a customized scrolling function.

```
typedef void TXNScrollInfoProcPtr (
      SInt32 iValue,
      SInt32 iMaximumValue,
      TXNScrollBarOrientation iScrollBarOrientation,
      SInt32 iRefCon
);
```

If you name your function `MyTXNScrollInfoProc`, you would declare it like this:

```
void MyTXNScrollInfoProc (
      SInt32 iValue,
      SInt32 iMaximumValue,
      TXNScrollBarOrientation iScrollBarOrientation,
      SInt32 iRefCon
);
```

Parameters

iValue
The scroll bar value.

iMaximumValue
The scroll bar maximum value.

iScrollBarOrientation
The orientation of the scroll bar. See Scroll Bar Orientation (page 147) for more information.

iRefCon
An unsigned 32-bit integer your application can use as needed. This is set and used by your application as needed.

Discussion
You can create a callback to handle and draw your own scroll bar. MLTE calls your callback each time the scroll bar value and the maximum value of the scroll bar needs to be updated. You install a text-scrolling callback on an MLTE text object by calling the function `TXNRegisterScrollInfoProc` (page 68). When you no longer want the text-scrolling callback invoked on the text object, you can call the function `TXNRegisterScrollInfoProc` with the `iTXNScrollInfoUPP` set to `NULL`.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`MacTextEditor.h`

Data Types

TXNActionNameMapperUPP

Defines a universal procedure pointer to an action name mapper callback function.

```
typedef TXNScrollActionNameProcPtr TXNActionNameMapperUPP;
```

Discussion
See `TXNActionNameMapperProcPtr` (page 89) for more information.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`MacTextEditor.h`

TXNActionKeyMapperUPP

Defines a universal procedure pointer to an action key mapping callback function.

```
typedef TXNActionKeyMapperProcPtr TXNActionKeyMapperUPP;
```

Discussion
See `TXNActionKeyMapperProcPtr` (page 88) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNATSUIFeatures

Contains information about ATSUI font features.

```
struct TXNATSUIFeatures {
    ItemCount featureCount;
    ATSUFontFeatureType * featureTypes;
    ATSUFontFeatureSelector * featureSelectors;
};
typedef struct TXNATSUIFeatures TXNATSUIFeatures;
```

Fields
`featureCount`
The number of features described in this structure.

`featureTypes`
: A pointer to a variable of type `ATSUFontFeatureType`. The `ATSUFontFeatureType` type represents the attributes of a particular font feature, such as the presence of ligatures in a font.

`featureSelectors`
: A pointer to a variable of type `ATSUFontFeatureSelector`. The `ATSUFontFeatureSelector` type represents the state of a feature (on or off).

Discussion
Used in the `TXNAttributeData` (page 94) union.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNATSUIVariations

Contains information about ATSUI variations.

```
struct TXNATSUIVariations {
    ItemCount variationCount;
    ATSUFontVariationAxis * variationAxis;
    ATSUFontVariationValue * variationValues;
};
typedef struct TXNATSUIVariations TXNATSUIVariations;
```

Fields

`variationCount`
: The number of variables described in this structure.

`variationAxis`
: A pointer to a variable of type `ATSUFontVariationAxis`. The `ATSUFontVariationAxis` type represents a stylistic attribute and the range of values used to express this attribute for a font.

`variationValues`
: A pointer to a variable of type `ATSUFontVariationValue`. The `ATSUFontVariationValue` type represents the range of values that a font can use for a particular font variation.

Discussion
Used in the `TXNAttributeData` (page 94) union.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNAttributeData

Contains information about text attributes in a text object.

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

Fields

`dataPtr`
: A pointer to attribute data. For example, a pointer to a font name.

`dataValue`
: A value that specifies a text attribute. For example, a value that specifies a font style.

`atsuFeatures`
: A pointer to a `TXNATSUIFeatures` (page 93) structure. For example, a structure that contains information about the ligatures in a font.

`atsuVariations`
: A pointer to a `TXNATSUIVariations` (page 94) structure. For example, a structure that contains information about the range of values used to express a particular stylistic attribute.

`urlReference`
: A URL that specifies the location of attribute data.

Discussion

Used in the `TXNTypeAttributes` (page 105) structure. The data contained in the union is determined by the value in the `size` field of the `TXNTypeAttributes` structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MacTextEditor.h`

TXNBackground

Specifies the background for text and other data in a text object.

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

Fields

`bgType`
: Defines the type of data. See `TXNBackgroundType` data type.

`bg`
: Specifies the data MLTE should use as a background.

Discussion

Used in the `TXNSetBackground` (page 72) function.

Version Notes

MLTE 1.1 and earlier supports only color.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNBackgroundData

Represents background data used in the `TXNBackground` structure.

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

Fields

`color`
: A value that specifies the background color on which data is displayed. Color is the only background data that is currently supported.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNCarbonEventInfo

Contains information needed for MLTE to support Carbon events.

```
struct TXNCarbonEventInfo {
    Boolean useCarbonEvents;
    UInt8 filler;
    UInt16 flags;
    CFDictionaryRef fDictionary;
};
typedef struct TXNCarbonEventInfo TXNCarbonEventInfo;
```

Fields

`useCarbonEvents`
: Pass `true` to specify a Carbon event, otherwise Apple events will be used by the system to handle MLTE events.

`filler`
: Padding you don't need to pass anything.

`flags`
: There are currently two flags defined: `kTXNNoAppleEventHandlersMask` and `kTXNRestartAppleEventHandlersMask`. When you request Carbon event support for text input events, it's best to specify the `kTXNNoAppleEventHandlersMask` mask.

`fDictionary`

A reference to a Core Foundation dictionary. (A dictionary is a collection of key-value pairs.) For MLTE to support Carbon events, you need to build a dictionary whose keys are strings that represent the events you want handled (such as "TextInput" or "WindowResize") and whose values are event target references associated with the events. See "Dictionary Keys" for a list of the predefined keys you can use to build the dictionary.

Discussion

You set up MLTE to support Carbon events by passing the `TXNCarbonEventInfo` structure when you call the function `TXNSetTXNObjectControls` (page 81). Note that MLTE does not handle subclass Carbon events dispatched by the standard handler unless you install the standard handler on the window by calling the Carbon Event Manager function `InstallStandardEventHandler` or by writing code that performs the tasks done by `InstallStandardEventHandler`.

MLTE supports four classes of Carbon events:

- `kEventClassTextInput`
- `kEventClassWindow`
- `kEventClassCommand`
- `kEventClassMenu`

The event kinds supported within each class are listed in Table 1. You don't need to specify event kinds to MLTE; the table is provided so you can see the kind of events that MLTE supports. When you set up support for Carbon events, MLTE handles all the calls to the Carbon Event Manager that actually install and set up the handlers that take care of the Carbon events for a text object. See *Inside Mac OS X: Handling Carbon Events* and the *Carbon Event Manager Reference* for more information on Carbon events and for the most recent list of event classes and kinds supported by MLTE.

Table 1 Event classes and kinds supported by MLTE

Event Class	Event Kind	Means
`kEventClassTextInput`	`kEventUnicodeForKeyEvent`	Text characters produced by a keyboard
	`kEventOffsetToPos`	Map character index to screen position
	`kEventPosToOffset`	Map screen position to character index
	`kEventGetSelectedText`	Determine currently selected text
	`kEventUpdateActiveInputArea`	Manage contents of an inline input session
`kEventClassWindow`	`kEventWindowActivated`	Window activated (brought to front)
	`kEventWindowDeactivated`	Window deactivated (sent behind)
	`kEventWindowDrawContent`	Draw window's contents on screen
	`kEventWindowClickContentRgn`	Mouse click in content region
`kEventClassCommand`	`kEventCommandProcess`	Menu item chosen or a control with a command has been pressed

	`kEventCommandUpdateStatus`	Determine enabled or disabled status of command
`kEventClassMenu`	`kEventMenuEnableItems`	Font selected from a Font menu

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNContextualMenuSetupUPP

Defines a universal procedure pointer to a contextual menu setup callback function.

```
typedef TXNContextualMenuSetupProcPtr TXNContextualMenuSetupUPP;
```

Discussion
See `TXNContextualMenuSetupProcPtr` (page 90) for more information.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`MacTextEditor.h`

TXNControlData

Contains information about formatting and other settings that control how a text object is displayed or behaves.

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

Fields

`uValue`
: A control setting. You should use this field for control settings, except tab and margins, whose value is unsigned.

`sValue`
: A control setting. You should use this only for control settings, except tab and margins, whose value is signed. There currently are no control settings whose value is signed.

`tabValue`
: A structure that contains tab distance and tab type settings.

`marginsPtr`
: A pointer to a `TXNMargins` (page 101) structure that specifies the top, left, and right margin settings.

Discussion
The `TXNControlData` data type is a parameter in the `TXNSetTXNObjectControls` (page 81) and `TXNGetTXNObjectControls` (page 59) functions. The data contained in the union is determined by the `iControlData` parameter of those functions.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNErrors

Defines result codes. (**Deprecated.** Use `OSStatus` instead.)

```
typedef OSStatus TXNErrors;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNFindUPP

Defines a universal procedure pointer to a find callback function.

```
typedef TXNFindProcPtr TXNFindUPP;
```

Discussion
See `TXNFindProcPtr` (page 90) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNFontMenuObject

Contains private variables necessary to represent an MLTE Font menu.

```
typedef struct OpaqueTXNFontMenuObject * TXNFontMenuObject;
```

Discussion
Used in the functions `TXNNewFontMenuObject` (page 185), `TXNPrepareFontMenu` (page 186), `TXNGetFontMenuHandle` (page 185), `TXNDoFontMenuSelection` (page 184), and `TXNDisposeFontMenuObject` (page 183).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNFrameID

Represents the text frame to which actions should be applied.

```
typedef UInt32 TXNFrameID;
```

Discussion
Used in the functions `TXNNewObject` (page 170), `TXNActivate` (page 164), `TXNResizeFrame` (page 69), `TXNSetFrameBounds` (page 77), `TXNDragReceiver` (page 38), and `TXNDragTracker` (page 39).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNLongRect

Contains coordinates for a view or text rectangle. (**Deprecated.** No longer needed.)

```
struct TXNLongRect {
    SInt32 top;
    SInt32 left;
    SInt32 bottom;
    SInt32 right;
};
typedef struct TXNLongRect TXNLongRect;
```

Fields

`top`
: The top coordinate of the rectangle.

`left`
: The left-side coordinate of the rectangle.

`bottom`
: The bottom coordinate of the rectangle.

`right`
: The right-side coordinate of the rectangle.

Special Considerations

The `TXNLongRect` data structure is passed as a parameter to the functions `TXNSetRectBounds` (page 174) and `TXNGetRectBounds` (page 167), which are deprecated. You should instead use the functions `TXNSetHIRectBounds` (page 78) and `TXNGetHIRect` (page 54), which take an `HIRect` data structure as a parameter instead of a `TXNLongRect` data structure

Availability
Available in Mac OS X v10.1 and later.

Declared In
`MacTextEditor.h`

TXNMacOSPreferredFontDescription

Contains information about the preferred font, font size, and style for a given text encoding.

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

Fields

`fontID`
: The ID of the preferred font.

`pointSize`
: The point size of the preferred font.

`encoding`
: The text encoding of the preferred font.

`fontStyle`
: The font style of the preferred font.

Discussion

Used in the functions `TXNInitTextension` (page 62), `TXNSetFontDefaults` (page 183), and `TXNGetFontDefaults` (page 179).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNMargins

Contains the margin values of a text object.

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

Fields

`topMargin`
: The location of the top margin. Available in MLTE version 1.2 and later.

`leftMargin`
: The location of the left margin. Available in MLTE version 1.2 and later.

`bottomMargin`
: The location of the bottom margin. This is a placeholder; it is currently not possible to set the bottom margin.

`rightMargin`
: The location of the right margin. Available in MLTE version 1.2 and later.

Discussion
This structure is used as a field in the `TXNControlData` (page 98) union.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNMatchTextRecord

Contains information about the text to be matched in a find operation.

```
struct TXNMatchTextRecord {
    const void * iTextPtr;
    SInt32 iTextToMatchLength;
    TextEncoding iTextEncoding;
};
typedef struct TXNMatchTextRecord TXNMatchTextRecord;
```

Fields

`iTextPtr`
: A pointer to the text to be matched.

`iTextToMatchLength`
: The length of text to which the `iTextPtr` parameter points.

`iTextEncoding`
: The encoding used by the text to be matched.

Discussion
Used in the `TXNFind` (page 45) function and the callback `TXNFindProcPtr` (page 90).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNObject

Contains private variables and functions necessary to represent text and handle text formatting at a document level.

```
typedef struct OpaqueTXNObject * TXNObject;
```

Discussion
You obtain a structure of type `TXNObject` from the `TXNNewObject` (page 170) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNObjectRefCon

Contains data specific to your application. (**Deprecated.** Used only in the `TXNNewObject` (page 170) function, which is deprecated.)

```
typedef void * TXNObjectRefCon;
```

Declared In
`MacTextEditor.h`

TXNScrollInfoUPP

Defines a universal procedure pointer to a scroll callback function.

```
typedef TXNScrollInfoProcPtr TXNScrollInfoUPP;
```

Discussion
See `TXNScrollInfoProcPtr` (page 92) for more information.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`MacTextEditor.h`

TXNTab

Contains tab information for a text object.

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

Fields

`value`
The distance between tabs.

`tabType`
The type of tab settings, such as right or left. See Tab Types (page 154) for a description of possible values.

`filler`
An unsigned 8-bit integer that exists only to make the structure exactly 4 bytes in size.

Discussion

Passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 81) function when the value of the `iControlTags` parameter is `kTXNTabSettingsTag`, or returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 59) function when the value of the `iControlTags` parameter is `kTXNTabSettingsTag`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MacTextEditor.h`

TXNTextBoxOptionsData

Contains information about how text appears in a Unicode text box.

```
struct TXNTextBoxOptionsData {
    TXNTextBoxOptions optionTags;
    Fract flushness;
    Fract justification;
    Fixed rotation;
    void * options;
};
typedef struct TXNTextBoxOptionsData TXNTextBoxOptionsData;
```

Fields

`optionTags`

Specifies the field in this structure at which MLTE should look. See Text Box Options Masks (page 157) for a description of possible values. You must supply the data associated with this tag in the appropriate field. For example, if you set the value of the `optionTags` field to `kTXNSetJustificationMask`, you must specify the type of justification in the `justification` field.

`flushness`

Indicates whether text should be displayed flush left, flush right, or centered in the text box. You should use one of the line justification constants defined in ATSUnicode.h. The possible values are `kATSUStartAlignment`, `kATSUEndAlignment`, and `kATSUCenterAlignment`.

`justification`

The type of justification to use in the text box. You should use one of the line justification constants defined in ATSUnicode.h. The possible values are `kATSUNoJustification` and `kATSUFullJustification`.

`rotation`

The angle of rotation for text in the text box.

`options`

Reserved for future use. This should be set to `NULL`.

Discussion

Used in the `TXNDrawUnicodeTextBox` (page 42) and `TXNDrawCFStringTextBox` (page 41) functions.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNTypeAttributes

Contains information about attributes for a text object.

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

Fields

`tag`

A value that specifies the type of information contained in the `data` field. See Font Run Attributes (page 126) for a description of possible values.

`size`

The size of the attribute. See Font Run Attribute Sizes (page 127) for a description of possible values.

`data`

A union that serves either as a 32-bit integer or a 32-bit pointer, depending on the `size` field.

Discussion

Used in the functions `TXNSetTypeAttributes` (page 82), `TXNGetContinuousTypeAttributes` (page 50), and `TXNGetIndexedRunInfoFromRange` (page 55).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MacTextEditor.h`

TXNVersionValue

Specifies the version of MLTE in use.

```
typedef UInt32 TXNVersionValue;
```

Discussion

Returned by the `TXNVersionInformation` (page 85) function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MacTextEditor.h`

TXTNTag

Specifies the type of information you want passed in the `data` field of the `TXNTypeAttributes` structure. (**Deprecated.** Use `TXNTypeRunAttributes` (page 126).)

```
typedef FourCharCode TXTNTag;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

Constants

Action Constants

Specify constants for actions used when calling `TXNCanUndoAction` (page 31) and `TXNCanRedoAction` (page 30).

```
const CFStringRef kTXNActionTyping;
const CFStringRef kTXNActionCut;
const CFStringRef kTXNActionPaste;
const CFStringRef kTXNActionClear;
const CFStringRef kTXNActionChangeFont;
const CFStringRef kTXNActionChangeColor;
const CFStringRef kTXNActionChangeSize;
const CFStringRef kTXNActionChangeStyle;
const CFStringRef kTXNActionAlignLeft;
const CFStringRef kTXNActionAlignCenter;
const CFStringRef kTXNActionAlignRight;
const CFStringRef kTXNActionDrop;
const CFStringRef kTXNActionMove;
const CFStringRef kTXNActionChangeFontFeature;
const CFStringRef kTXNActionChangeFontVariation;
const CFStringRef kTXNActionChangeGlyphVariation;
const CFStringRef kTXNActionChangeTextPosition;
const CFStringRef kTXNActionUndoLast;
```

Constants

`kTXNActionTyping`
A typing action.

`kTXNActionCut`
A cut action.

`kTXNActionPaste`
A paste action.

`kTXNActionClear`
A clear action.

`kTXNActionChangeFont`
A font change action.

`kTXNActionChangeColor`
A color change action.

`kTXNActionChangeSize`
A size change action.

`kTXNActionChangeStyle`
A change in style action.

`kTXNActionAlignLeft`
An align left action.

`kTXNActionAlignCenter`
An align center action.

`kTXNActionAlignRight`
An align right action.

`kTXNActionDrop`
A drop action.

`kTXNActionMove`
A move action.

`kTXNActionChangeFontFeature`
A change font feature action.

`kTXNActionChangeFontVariation`
A change in font variation action.

`kTXNActionChangeGlyphVariation`
A change glyph variation action.

`kTXNActionChangeTextPosition`
A change text position action; includes changing the space before and after characters and shifting the text's baseline.

`kTXNActionUndoLast`
Used in undo and redo functions if none of the other constants apply.

Discussion
Use these constants when calling the `TXNCanUndoAction` (page 31) and `TXNCanRedoAction` (page 30) functions.

Declared In
`MacTextEditor.h`

Action Count Constants

Represent action types use by `TXNGetCountForActionType` (page 51) and `TXNClearCountForActionType` (page 32).

```
const CFStringRef kTXNActionCountOfTextChanges;
const CFStringRef kTXNActionCountOfStyleChanges;
const CFStringRef kTXNActionCountOfAllChanges;
```

Constants
`kTXNActionCountOfTextChanges`
Count of text changes. All text changes other than style changes and custom defined actions are included in this action count. Includes key presses, inline sessions, cut, copy, and paste, and drop. Undo and redo events of these kinds are also included in this action count.

`kTXNActionCountOfStyleChanges`
Count of text style changes. Style changes include changing font, font face, font size, font feature, font variation, font color, glyph variation, and text position. Undo or redo events of these kinds are also included in this action count.

`kTXNActionCountOfAllChanges`
Total count of actions including all text and style changes, as well as custom defined actions.

Declared In
`MacTextEditor.h`

Action Count Bits

Specify actions to be included in an action count when calling `TXNGetActionChangeCount` (page 179) and `TXNClearActionChangeCount` (page 178). (**Deprecated.** See Action Count Constants (page 107).)

```
enum {
    kTXNTextInputCountBit = 0,
    kTXNRunCountBit = 1
};
```

Constants

`kTXNTextInputCountBit`

When this bit is set, general text input events that affect the content of the document are included in the action count. General text input events include key presses, inline sessions, pasting, cutting, dropping, and other editing events. Undo or redo events of text input events are also included in the action count.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNRunCountBit`

When this bit is set, general style changes to the text are included in the action count. Style changes include changes to the text face, font, font size and so forth. Undo and redo events of style changes are also included in the action count.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

Declared In

`MacTextEditor.h`

Action Count Masks

Set or test action count bits for use with `TXNGetActionChangeCount` (page 179) and `TXNClearActionChangeCount` (page 178). (**Deprecated.** See Action Count Constants (page 107).)

```
typedef OptionBits TXNCountOptions;
enum {
    kTXNTextInputCountMask = 1L << kTXNTextInputCountBit,
    kTXNRunCountMask = 1L << kTXNRunCountBit,
    kTXNAllCountMask = kTXNTextInputCountMask | kTXNRunCountMask
};
```

Constants

`kTXNTextInputCountMask`

Use to set or test for the `kTXNTextInputCountBit`.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNRunCountMask`

Used to set or test for the `kTXNRunCountBit`.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNAllCountMask`

Use to set or text for both `kTXNTextInputCountBit` and `kTXNRunCountBit`.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

Declared In

`MacTextEditor.h`

Action Types

Specify constants for editing actions taken by the user in versions of Mac OS X prior to Mac OS X v10.4. (**Deprecated.** These constants were used in the `TXNCanUndo` (page 177) and `TXNCanRedo` (page 177) functions, which are deprecated in Mac OS X v10.4. Use the `TXNCanUndoAction` (page 31) and `TXNCanRedoAction` (page 30) functions instead.)

```
typedef UInt32 TXNActionKey;
enum {
    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,
    kTXNFontFeatureAction = 13,
    kTXNFontVariationAction = 14,
    kTXNUndoLastAction = 1024
};
```

Constants

`kTXNTypingAction`

A typing action.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNCutAction`

A cut action.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNPasteAction`

A paste action.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNClearAction`
: A clear action.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNChangeFontAction`
: A font change action.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNChangeFontColorAction`
: A change in font color action.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNChangeFontSizeAction`
: A change in font size action.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNChangeStyleAction`
: A change in font style action.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNAlignLeftAction`
: An align left action.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNAlignCenterAction`
: An align center action.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNAlignRightAction`
: An align right action.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNDropAction`
: A drop action.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNMoveAction`
: A move selection action.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNFontFeatureAction`

A change in font feature action.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNFontVariationAction`

A change in font variation action.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNUndoLastAction`

Use this if none of the other constants apply.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Declared In

`MacTextEditor.h`

Apple Event Handler Bits

Specify whether Apple events should be used.

```
enum {
    kTXNNoAppleEventHandlersBit = 0,
    kTXNRestartAppleEventHandlersBit = 1
};
```

Constants

`kTXNNoAppleEventHandlersBit`

When this bit is set, Apple event handlers are not used. (**Deprecated.** There is no replacement.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNRestartAppleEventHandlersBit`

When this bit is set, Apple event handlers are started up. (**Deprecated.** There is no replacement.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Special Considerations

MLTE does not support Apple events in Mac OS X version 10.1 and later.

Declared In

`MacTextEditor.h`

Apple Event Handler Masks

Set or test Apple event handler bits.

```
enum {
    kTXNNoAppleEventHandlersMask = 1 << kTXNNoAppleEventHandlersBit,
    kTXNRestartAppleEventHandlersMask = 1 << kTXNRestartAppleEventHandlersBit
};
```

Constants

`kTXNNoAppleEventHandlersMask`

Use to set or test for the `kTXNNoAppleEventHandlersBit`. Use this if you want Apple event handlers removed. (**Deprecated.** There is no replacement.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNRestartAppleEventHandlersMask`

Used to set or test for the `kTXNRestartAppleEventHandlersBit`. Use this if you want to subsequently restart Apple event handlers after removing your own text handlers. (**Deprecated.** There is no replacement.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion

These constants are currently the only settings for the flags field of `TXNCarbonEventInfo`.

Special Considerations

MLTE does not support Apple events in Mac OS X version 10.1 and later

Declared In

`MacTextEditor.h`

ATSUI Feature Bits

Specify the default imaging system.

```
enum {
    kTXNWillDefaultToATSUIBit = 0,
    kTXNWillDefaultToCarbonEventBit = 1
};
```

Constants

`kTXNWillDefaultToATSUIBit`

When this bit is set, indicates ATSUI is the default imaging system.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNWillDefaultToCarbonEventBit`

When this bit is set, indicates MLTE uses Carbon events by default.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

Declared In

`MacTextEditor.h`

ATSUI Feature Masks

Specifies new features and represents the default imaging system.

```
typedef OptionBits TXNFeatureBits;
enum {
    kTXNWillDefaultToATSUIMask = 1L << kTXNWillDefaultToATSUIBit,
    kTXNWillDefaultToCarbonEventMask = 1L << kTXNWillDefaultToCarbonEventBit
};
```

Constants

`kTXNWillDefaultToATSUIMask`

Test for ATSUI as the default imaging system.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNWillDefaultToCarbonEventMask`

Test for Carbon events as the default event handling mechanism.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

Discussion

You can use this to test for bit 0 in the `oFeatureFlags` parameter returned by the `TXNVersionInformation` (page 85) function.

Declared In

`MacTextEditor.h`

Automatic Indentation Settings

Specify the automatic indentation setting for a text object.

```
enum {
    kTXNAutoIndentOff = false,
    kTXNAutoIndentOn = true
};
```

Constants

`kTXNAutoIndentOff`

Automatic indenting is not enabled.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNAutoIndentOn`

Automatic indenting is enabled. You can enable this feature only if automatic word wrapping is not enabled.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion

Passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 81) function when the value of the `iControlTags` parameter is `kTXNAutoIndentStateTag`, or returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 59) function when the value of the `iControlTags` parameter is `kTXNAutoIndentStateTag`.

Declared In
`MacTextEditor.h`

Automatic Scrolling Behavior

Specify automatic scrolling behavior for a text object.

```
enum {
    kTXNAutoScrollInsertionIntoView = 0,
    kTXNAutoScrollNever = 1,
    kTXNAutoScrollWhenInsertionVisible = 2
};
typedef UInt32 TXNAutoScrollBehavior;
```

Constants

`kTXNAutoScrollInsertionIntoView`
: The default auto scrolling behavior. When text is inserted, the document is scrolled to show the new insertion. This was the only type of autoscrolling prior to Mac OS X v10.4.

  Available in Mac OS X v10.4 and later.

  Declared in `MacTextEditor.h`.

`kTXNAutoScrollNever`
: Never autoscroll, even when dragging the mouse or inserting text. The only way to scroll the document is for the user to use the scrollbar or to scroll programatically.

  Available in Mac OS X v10.4 and later.

  Declared in `MacTextEditor.h`.

`kTXNAutoScrollWhenInsertionVisible`
: Autoscrolling only happens when the insertion offset is currently in the user's view. If the user is looking at the first page of a ten page document and text is inserted at the end of the document, no autoscrolling occurs. However, if the user was looking at page ten and text is inserted there, the document would scroll. This type of autoscrolling is best for implementing terminal or log windows.

  Available in Mac OS X v10.4 and later.

  Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Carbon Event Dictionary Keys

Specify Carbon events to be handled by MLTE.

```
#define kTXNTextHandlerKey CFSTR("TextInput")
#define kTXNWindowEventHandlerKey CFSTR("WindowEvent")
#define kTXNWindowResizeEventHandlerKey CFSTR("WindowResize")
#define kTXNCommandTargetKey CFSTR("CommandTarget")
#define kTXNCommandUpdateKey CFSTR("CommandUpdate")
#define kTXNFontMenuObjectKey CFSTR("FontMenuObject")
#define kTXNActionNameMapperKey CFSTR("ActionNameMapper")
#define kTXNWheelMouseEventHandlerKey CFSTR("WheelMouseEvent")
#define kTXNTSMDocumentAccessHandlerKey CFSTR("TSMDocumentAccess")
#define kTXNFontPanelEventHandlerKey CFSTR("FontPanel")
#define kTXNFontMenuRefKey CFSTR("FontMenuRef")
#define kTXNActionKeyMapperKey CFSTR("ActionKeyMapper")
```

Constants

`kTXNTextHandlerKey`
Indicates the Carbon event class `kEventClassTextInput` and the event kinds `kEventTextInputUpdateActiveInputArea`, `kEventTextInputUnicodeForKeyEvent`, `kEventTextInputOffsetToPos`, `kEventTextInputPosToOffset`, and `kEventTextInputGetSelectedText`.

`kTXNWindowEventHandlerKey`
Indicates the Carbon event class `kEventClassWindow` and the event kinds `kEventWindowActivated`, `kEventWindowDeactivated`, `kEventWindowDrawContent`, and `kEventWindowClickContentRegion`.

`kTXNWindowResizeEventHandlerKey`
Indicates the Carbon event class `kEventClassWindow` and a window resizing event.

`kTXNCommandTargetKey`
Indicates the Carbon event class `kEventClassCommand` and the event kind `kEventCommandProcess`.

`kTXNCommandUpdateKey`
Indicates the Carbon event class `kEventClassCommand` and the event kind `kEventCommandUpdate`.

`kTXNFontMenuObjectKey`
Indicates the Carbon event class `kEventClassMenu` and the event kind `kEventMenuEnableItems`.

`kTXNActionNameMapperKey`
Indicates an action key mapper callback function. Available in Mac OS X v10.4; use instead of `kTXNActionKeyMapperKey`.

`kTXNWheelMouseEventHandlerKey`
Indicates the handler for wheel mouse events.

`kTXNTSMDocumentAccessHandlerKey`
Indicates the handler for TSM document access events.

`kTXNFontPanelEventHandlerKey`
Indicates the handler for Font Panel events.

`kTXNFontMenuRefKey`
Indicates the Carbon event class `kEventClassMenu`.

`kTXNActionKeyMapperKey`
Indicates an action key mapper callback function. (**Deprecated.** Use `kTXNActionNameMapperKey` instead.)

Declared In
`MacTextEditor.h`

Clearance Settings

Clear formatting and privileges settings.

```
enum {
    kTXNClearThisControl = 0xFFFFFFFF,
    kTXNClearTheseFontFeatures = 0x80000000
};
```

Constants

`kTXNClearThisControl`

Clears control settings. If you want to clear a setting associated with a control tag, you can call the `TXNSetTXNObjectControls` (page 81) function with the value of the `iControlData` parameter set to `kTXNClearThisControl`. MLTE resets the value of the control specified in the `iControlTag` parameter of the `TXNSetTXNObjectControls` (page 81) function to the default value for that control.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNClearTheseFontFeatures`

Clears font feature settings. You can use this constant when you call the `TXNSetTypeAttributes` (page 82) function to clear all of the ATSUI font features or ATSUI font variations.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Declared In

`MacTextEditor.h`

Command Event Support Options

Specify options for enabling support for command events in a TXNObject.

```
enum {
    kTXNSupportEditCommandProcessing = 1 << 0,
    kTXNSupportEditCommandUpdating = 1 << 1,
    kTXNSupportSpellCheckCommandProcessing = 1 << 2,
    kTXNSupportSpellCheckCommandUpdating = 1 << 3,
    kTXNSupportFontCommandProcessing = 1 << 4,
    kTXNSupportFontCommandUpdating = 1 << 5
};
typedef UInt32 TXNCommandEventSupportOptions;
```

Constants

`kTXNSupportEditCommandProcessing`

Setting this bit when calling `TXNSetTXNObjectControls` (page 81) enables support for processing the menu item associated with `kHICommandUndo`, `kHICommandRedo`, `kHICommandCut`, `kHICommandCopy`, `kHICommandPaste`, `kHICommandClear`, and `kHICommandSelectAll`. If this bit is not set when `TXNSetTXNObjectControls` (page 81) is called, support is disabled.

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

`kTXNSupportEditCommandUpdating`

Setting this bit when calling `TXNSetTXNObjectControls` (page 81) enables support for updating the menu item associated with `kHICommandUndo`, `kHICommandRedo`, `kHICommandCut`, `kHICommandCopy`, `kHICommandPaste`, `kHICommandClear`, and `kHICommandSelectAll`. For Undo, the item is enabled if there are any undoable actions in MLTE's command stack, and, if you have installed an action key mapper proc, it is called to get the appropriate string for the Undo item. For Redo, the item is enabled if there are any redoable actions; if you have installed an action key mapper callback, it is called to get the appropriate string for the Redo item. For Cut and Clear, the item is enabled if there is current selection that is not empty; otherwise, these items are disabled. For Paste, the item is enabled if the clipboard is not empty; it is disabled if the clipboard is empty or contains data that MLTE cannot parse. For Select All, the item is always updated. If this bit is not set when `TXNSetTXNObjectControls` (page 81) is called, support is disabled.

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

`kTXNSupportSpellCheckCommandProcessing`

Setting this bit when calling `TXNSetTXNObjectControls` (page 81) enables support for spell checking. The spell checking commands supported are: Show Spelling Panel (`'shsp'`), Check Spelling (`'cksp'`), Change Spelling (`'chsp'`), enable check spelling as you type (`'aspc'`), ignore spelling (`'igsp'`), and learn spelling (`'lrwd'`). If this bit is not set when `TXNSetTXNObjectControls` (page 81) is called, support is disabled.

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

`kTXNSupportSpellCheckCommandUpdating`

Enables support for updating the menu item associated with a given spell checking command. Once `kTXNSupportSpellCheckCommandUpdating` is enabled, the Show Spelling and Check Spelling items are always enabled. The Change Spelling item is included in a spelling menu only if the current selection is a misspelled word; it is disabled if the current selection is empty or not a misspelled word. The Check Spelling as You Type item is always enabled. It is checked if this feature has been enabled. By default when you turn on spell checking, this item is enabled. If this feature has been disabled, the item is not checked. Ignore Spelling usually does not have a corresponding menu item. If a menu does have this item, Ignore Spelling is disabled if the current selection is empty or is not a misspelled word. It is enabled if the current selection is a misspelled word. Learn Spelling typically does not have a corresponding menu item. If a menu does have this item, Learn Spelling is disabled if the current selection is empty or is not a misspelled word. It is enabled if the current selection is a misspelled word. If this bit is not set when `TXNSetTXNObjectControls` (page 81) is called, support is disabled.

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

`kTXNSupportFontCommandProcessing`

Setting this bit enables Carbon Font Panel support. Once enabled, MLTE handles the following Carbon Events defined in `FontPanel.h`: `kHICommandShowHideFontPanel` and `kEventFontPanelClosed` to show and hide the Carbon font panel, `kEventFontSelection` event to update the document after the selection of a new font, size, style, color, or any feature settings from the Typography Panel. If this bit is not set when `TXNSetTXNObjectControls` (page 81) is called, support is disabled.

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

`kTXNSupportFontCommandUpdating`
: Setting this bit enables support for updating the selection in Carbon Font Panel when the current selection in an MLTE document is changed. When this bit is set, `kTXNSupportFontCommandProcessing` must also be set. If this bit is not set when `TXNSetTXNObjectControls` (page 81) is called, support is disabled.

  Available in Mac OS X v10.4 and later.

  Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Continuous Style Information Bits

Specify whether font information is continuous.

```
enum {
    kTXNFontContinuousBit = 0,
    kTXNSizeContinuousBit = 1,
    kTXNStyleContinuousBit = 2,
    kTXNColorContinuousBit = 3
};
```

Constants

`kTXNFontContinuousBit`
: When this bit is set, the font is continuous in a text run.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNSizeContinuousBit`
: When this bit is set, the font size is continuous in a text run.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNStyleContinuousBit`
: When this bit is set, the font style is continuous in a text run.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNColorContinuousBit`
: When this bit is set, the font color is continuous in a text run.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Continuous Style Information Masks

Represents continuous style information needed by your application.

```
typedef OptionBits TXNContinuousFlags;
enum {
    kTXNFontContinuousMask = 1L << kTXNFontContinuousBit,
    kTXNSizeContinuousMask = 1L << kTXNSizeContinuousBit,
    kTXNStyleContinuousMask = 1L << kTXNStyleContinuousBit,
    kTXNColorContinuousMask = 1L << kTXNColorContinuousBit
};
```

Constants

`kTXNFontContinuousMask`

Use to test for continuous font information in a text run.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNSizeContinuousMask`

Use to test for continuous size information in a text run.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNStyleContinuousMask`

Use to test for continuous style information in a text run.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNColorContinuousMask`

Use to test for continuous color information in a text run.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion

Used in the `TXNGetContinuousTypeAttributes` (page 50) function.

Declared In

`MacTextEditor.h`

Data Offsets

Specifies offsets to use when manipulating data in a text object. Offsets in MLTE are always character offsets.

```
typedef UInt32 TXNOffset;
enum {
    kTXNUseCurrentSelection = 0xFFFFFFFF,
    kTXNStartOffset = 0,
    kTXNEndOffset = 0x7FFFFFFF
};
```

Constants

`kTXNUseCurrentSelection`

Use the current selection.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNStartOffset`
: The first offset of the text in a text object.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNEndOffset`
: The last offset of the text in a text object.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

Discussion

These constants can be passed and returned in functions that have the parameter of type `TXNOffset`.

Declared In

`MacTextEditor.h`

Data Option Key Value Constants

Specifies constants used as key values in Data Option dictionaries.

```
const CFStringRef kTXNPlainTextDocumentType
const CFStringRef kTXNMLTEDocumentType
const CFStringRef kTXNRTFDocumentType
const CFStringRef kTXNQuickTimeDocumentType
```

Constants

`kTXNPlainTextDocumentType`
: Plain text document.

`kTXNMLTEDocumentType`
: Native MLTE document type.

`kTXNRTFDocumentType`
: Rich text format (RTF) document type.

`kTXNQuickTimeDocumentType`
: Multimedia file that can be opened by QuickTime importers. This document type is only supported for reading data, not for writing data.

Discussion

These constants are passed in Data Option dictionary keys to `TXNReadFromCFURL` (page 67) and `TXNWriteRangeToCFURL` (page 86) to specify options that are to be used when reading data into a TXNObject and when writing data from a TXNObject to a file or special file bundle (directory).

Declared In

`MacTextEditor.h`

Data Option Key Constants

Specifies keys for use in dictionaries passed as parameters to `TXNReadFromCFURL` (page 67) and `TXNWriteRangeToCFURL` (page 86).

```
    const CFStringRef kTXNDataOptionDocumentTypeKey
    const CFStringRef kTXNDataOptionCharacterEncodingKey
```

Constants

`kTXNDataOptionDocumentTypeKey`

`CFString` specifying the document format. The following constants are supported: `kTXNPlainTextDocumentType`, `kTXNMLTEDocumentType`, `kTXNRTFDocumentType`, and `kTXNQuickTimeDocumentType`. For information on these constants, see Data Option Key Value Constants (page 120).

`kTXNDataOptionCharacterEncodingKey`

CFNumber of type `kCFNumberSInt32Type` containing the character encoding as specified in `CFString.h` and `CFStringEncodingExt.h`.

Discussion

Data options are used to specify options for reading in and writing out data.

Declared In

`MacTextEditor.h`

Default Font Name

Specifies the default font name.

```
enum {
    kTXNDefaultFontName = 0
};
```

Constants

`kTXNDefaultFontName`

The default font name.

Discussion

MLTE used these constants in an earlier version in which only a single font was allowed. You can now specify an array of font descriptions by using the `TXNMacOSPreferredFontDescription` (page 101) structure. See the function `TXNInitTextension` (page 62) for a description of how to specify defaults for a font.

Declared In

`MacTextEditor.h`

Default Font Size

Specifies the default font size.

```
enum {
    kTXNDefaultFontSize = 0x000C0000
};
```

Constants

`kTXNDefaultFontSize`

Sets default font size.

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Default Font Style

Specifies the default font style.

```
enum {
    kTXNDefaultFontStyle = normal
};
```

Constants

`kTXNDefaultFontStyle`
Sets default font style.

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Document Attribute Keys

Specify dictionary keys for document attribute dictionaries used by `TXNWriteRangeToCFURL` (page 86) and `TXNReadFromCFURL` (page 67).

```
const CFStringRef kTXNDocumentAttributeTitleKey;
const CFStringRef kTXNDocumentAttributeCompanyNameKey;
const CFStringRef kTXNDocumentAttributeSubjectKey;
const CFStringRef kTXNDocumentAttributeAuthorKey;
const CFStringRef kTXNDocumentAttributeKeywordsKey;
const CFStringRef kTXNDocumentAttributeCommentKey;
const CFStringRef kTXNDocumentAttributeEditorKey;
const CFStringRef kTXNDocumentAttributeCreationTimeKey;
const CFStringRef kTXNDocumentAttributeModificationTimeKey;
const CFStringRef kTXNDocumentAttributeCopyrightKey;
```

Constants

`kTXNDocumentAttributeTitleKey`
`CFString` containing the document's title.

`kTXNDocumentAttributeCompanyNameKey`
`CFString` containing the company name.

`kTXNDocumentAttributeSubjectKey`
`CFString` containing the document's subject.

`kTXNDocumentAttributeAuthorKey`
`CFString` containing the name of the document's author.

`kTXNDocumentAttributeKeywordsKey`
`CFArray` of values of type `CFString` containing keywords.

`kTXNDocumentAttributeCommentKey`
`CFString` containing comments.

`kTXNDocumentAttributeEditorKey`
: `CFString` containing the name of the person who last edited the document.

`kTXNDocumentAttributeCreationTimeKey`
: `CFAbsoluteTime` containing document comments; note that this is not the file system creation date of the file, but of the document, as it is stored in the document.

`kTXNDocumentAttributeModificationTimeKey`
: `CFAbsoluteTime` containing the last modification date of the document contents.

`kTXNDocumentAttributeCopyrightKey`
: `CFString` containing the copyright of the document.

Discussion

Use these constants when working with document attribute dictionaries that are passed to `TXNWriteRangeToCFURL` (page 86) and `TXNReadFromCFURL` (page 67).

When writing data out, document attributes are embedded into the data stream for document formats that support them (i.e. MLTE native format and RTF). When reading data in, document attributes are extracted from the data stream if the document format supports them.

Declared In

`MacTextEditor.h`

Drag and Drop Constants

Specify whether or not drag and drop is enabled.

```
enum {
    kTXNEnableDragAndDrop = false,
    kTXNDisableDragAndDrop = true
};
```

Constants

`kTXNEnableDragAndDrop`
: Enables drag and drop when passed as a parameter to the function `TXNSetTXNObjectControls` (page 81). Indicates drag and drop is disabled when returned from the function `TXNGetTXNObjectControls` (page 59).

  Available in Mac OS X v10.1 and later.

  Declared in `MacTextEditor.h`.

`kTXNDisableDragAndDrop`
: Disables drag and drop when passed as a parameter to the function `TXNSetTXNObjectControls` (page 81). Indicates drag and drop is enabled when returned from the function `TXNGetTXNObjectControls` (page 59).

  Available in Mac OS X v10.1 and later.

  Declared in `MacTextEditor.h`.

Declared In

`MacTextEditor.h`

Draw Items Bits

Specify which elements of the text object to render.

```
enum {
    kTXNDrawItemScrollbarsBit= 0,
    kTXNDrawItemTextBit= 1,
    kTXNDrawItemTextAndSelectionBit = 2
};
```

Constants

`kTXNDrawItemScrollbarsBit`

Specifies to draw the scroll bars.

Available in Mac OS X v10.3 and later.

Declared in `MacTextEditor.h`.

`kTXNDrawItemTextBit`

Specifies to render the text.

Available in Mac OS X v10.3 and later.

Declared in `MacTextEditor.h`.

`kTXNDrawItemTextAndSelectionBit`

Specifies to render the text and the current selection.

Available in Mac OS X v10.3 and later.

Declared in `MacTextEditor.h`.

Discussion

See `Draw Items Masks` (page 124).

Declared In

`MacTextEditor.h`

Draw Items Masks

Test for draw-items bits.

```
enum {
    kTXNDrawItemScrollbarsMask = 1UL << kTXNDrawItemScrollbarsBit,
    kTXNDrawItemTextMask = 1UL << kTXNDrawItemTextBit,
    kTXNDrawItemTextAndSelectionMask = 1UL <<
kTXNDrawItemTextAndSelectionBit,
    kTXNDrawItemAllMask =  0xFFFFFFFF
};
typedef OptionBits TXNDrawItems;
```

Constants

`kTXNDrawItemScrollbarsMask`

Use to set or test for the `kTXNDrawItemScrollbarsBit`.

Available in Mac OS X v10.3 and later.

Declared in `MacTextEditor.h`.

`kTXNDrawItemTextMask`

Used to set or test for the `kTXNDrawItemTextBit`.

Available in Mac OS X v10.3 and later.

Declared in `MacTextEditor.h`.

`kTXNDrawItemTextAndSelectionMask`
: Use to set or text for the `kTXNDrawItemTextAndSelectionBit`.

  Available in Mac OS X v10.3 and later.

  Declared in `MacTextEditor.h`.

`kTXNDrawItemAllMask`
: Used to set all draw-items bits or test to see whether all draw-items bits are set. Setting all bits specifies to draw the scroll bars, text, and the current selection.

  Available in Mac OS X v10.3 and later.

  Declared in `MacTextEditor.h`.

Discussion

These constants can be passed as parameters to the function `TXNDrawObject` (page 42).

Declared In

`MacTextEditor.h`

Font Defaults

Specify a variety of font settings.

```
enum {
    kTXNDontCareTypeSize = 0xFFFFFFFF,
    kTXNDontCareTypeStyle = 0xFF,
    kTXNIncrementTypeSize = 0x00000001,
    kTXNDecrementTypeSize = 0x80000000,
    kTXNUseScriptDefaultValue = -1,
    kTXNNoFontVariations = 0x7FFF
};
```

Constants

`kTXNDontCareTypeSize`
: Use the application default font size.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNDontCareTypeStyle`
: Use “normal” should as the font style.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNIncrementTypeSize`
: Increase the font size should by one point.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNDecrementTypeSize`
: Decrease the font size by one point.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

Discussion

These constants can be used as parameters in a variety of functions that control font attributes, such as the `TXNSetFontDefaults` (page 183) and `TXNSetTypeAttributes` (page 82) functions.

Declared In

`MacTextEditor.h`

Font Run Attributes

Specifies a font attribute (font family, size, style, and so forth) for a text run in a text object.

```
typedef FourCharCode TXNTypeRunAttributes;
enum {
    kTXNQDFontNameAttribute = 'fntn',
    kTXNQDFontFamilyIDAttribute = 'font',
    kTXNQDFontSizeAttribute = 'size',
    kTXNQDFontStyleAttribute = 'face',
    kTXNQDFontColorAttribute = 'klor',
    kTXNTextEncodingAttribute = 'encd',
    kTXNATSUIFontFeaturesAttribute = 'atfe',
    kTXNATSUIFontVariationsAttribute = 'atva',
    kTXNURLAttribute = 'urla'
    kTXNATSUIStyle  = 'astl'
};
```

Constants

`kTXNQDFontNameAttribute`

Specifies that the `data` field of the `TXNTypeAttributes` (page 105) structure contains a font name.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNQDFontFamilyIDAttribute`

Specifies that the `data` field of the `TXNTypeAttributes` (page 105) structure contains a font family ID.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNQDFontSizeAttribute`

Specifies that the `data` field of the `TXNTypeAttributes` (page 105) structure contains a font size. Obsolete; incorrect font sizes are always returned as a fixed value. (**Deprecated.** Use `kTXNFontSizeAttribute` instead.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNQDFontStyleAttribute`

Specifies that the `data` field of the `TXNTypeAttributes` (page 105) structure contains a font style.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNQDFontColorAttribute`
: Specifies that the `data` field of the `TXNTypeAttributes` (page 105) structure contains a font color.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNTextEncodingAttribute`
: Specifies that the `data` field of the `TXNTypeAttributes` (page 105) structure contains a text encoding.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNATSUIFontFeaturesAttribute`
: Specifies that the `data` field of the `TXNTypeAttributes` (page 105) structure contains ATSUI font features.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNURLAttribute`
: Specifies that the `data` field of the `TXNTypeAttributes` (page 105) structure contains a URL.

  Available in Mac OS X v10.2 and later.

  Declared in `MacTextEditor.h`.

`kTXNATSUIStyle`
: Specifies that the `data` field of the `TXNTypeAttributes` (page 105) structure contains an ATSUI style.

  Available in Mac OS X v10.3 and later.

  Declared in `MacTextEditor.h`.

Discussion

You pass these constants in the `tag` field of the `TXNTypeAttributes` (page 105) structure. You can supplement these with the style attributes defined for ATSUI.

Declared In

`MacTextEditor.h`

Font Run Attribute Sizes

Describes the size of a font attribute.

```
typedef ByteCount TXNTypeRunAttributeSizes;
enum {
    kTXNQDFontNameAttributeSize = sizeof(Str255),
    kTXNQDFontFamilyIDAttributeSize = sizeof(SInt16),
    kTXNQDFontSizeAttributeSize = sizeof(SInt16),
    kTXNQDFontStyleAttributeSize = sizeof(Style),
    kTXNQDFontColorAttributeSize = sizeof(RGBColor),
    kTXNTextEncodingAttributeSize = sizeof(TextEncoding),
    kTXNFontSizeAttributeSize = sizeof(Fixed),
    kTXNATSUIStyleSize = sizeof(ATSUStyle)
};
```

Constants

`kTXNQDFontNameAttributeSize`

The size of a QuickDraw font name.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNQDFontFamilyIDAttributeSize`

The size of a font family ID attribute.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNQDFontSizeAttributeSize`

Obsolete don't use. (**Deprecated.** Instead, use `kTXNFontSizeAttributeSize`.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNQDFontStyleAttributeSize`

The size of font style attribute.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNQDFontColorAttributeSize`

The size of a font color attribute.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNTextEncodingAttributeSize`

The size of text encoding attribute.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNFontSizeAttributeSize`

The size of the font size attribute. Use this instead of the `kTXNQDFontSizeAttributeSize` constant. Font sizes are always returned as a `Fixed` value.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNATSUIStyleSize`

The size of the ATSUI style attribute.

Available in Mac OS X v10.3 and later.

Declared in `MacTextEditor.h`.

Discussion

You pass these constants in the `size` field of the `TXNTypeAttributes` (page 105) structure.

Declared In

`MacTextEditor.h`

Formatting and Privileges Settings

Specifies the formatting and privileges information to get from or set for a text object.

```
typedef FourCharCode TXNControlTag;
enum {
    kTXNLineDirectionTag = 'lndr',
    kTXNJustificationTag = 'just',
    kTXNIOPrivilegesTag = 'iopv',
    kTXNSelectionStateTag = 'slst',
    kTXNInlineStateTag = 'inst',
    kTXNWordWrapStateTag = 'wwrs',
    kTXNKeyboardSyncStateTag = 'kbsy',
    kTXNAutoIndentStateTag = 'auin',
    kTXNTabSettingsTag = 'tabs',
    kTXNRefConTag = 'rfcn',
    kTXNMarginsTag = 'marg',
    kTXNFlattenMoviesTag = 'flat',
    kTXNDoFontSubstitution = 'fSub',
    kTXNNoUserIOTag = 'nuio',
    kTXNUseCarbonEvents = 'cbcb',
    kTXNDrawCaretWhenInactiveTag = 'dcrt',
    kTXNDrawSelectionWhenInactiveTag = 'dsln',
    kTXNDisableDragAndDropTag = 'drag',
    kTXNSingleLevelUndoTag = 'undo',
    kTXNVisibilityTag = 'visb'
    kTXNDisableLayoutAndDrawTag = kTXNVisibilityTag,
    kTXNAutoScrollBehaviorTag = ‘sbev’
};
```

Constants

`kTXNLineDirectionTag`

Specifies a setting for the direction text is written on the line. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 81) function, you must also pass a value that specifies the line direction in the `iControlData` parameter. See Line Direction Settings (page 144) for a description of possible values.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNJustificationTag`

Specifies a justification setting. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 81) function, you must also pass a value that specifies a justification setting in the `iControlData` parameter. See Justification Settings (page 142) for a description of possible values.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNIOPrivilegesTag`
: Indicates a privileges setting. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 81) function, you must also pass a value that specifies a privileges setting in the `iControlData` parameter. See Read and Write Privileges Settings (page 145) for a description of possible values.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNSelectionStateTag`
: Specifies a selection state; that is, whether MLTE displays a cursor and allows selections in a read-only document. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 81) function, you must also pass a value that specifies the selection state in the `iControlData` parameter. See Selection State Settings (page 151) for a description of possible values.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNInlineStateTag`
: Specifies an inline state that is, whether text is input through the document's window (inline) or through a small floating window that appears at the bottom of the screen. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 81) function, you must also pass a value that specifies the inline state in the `iControlData` parameter. See Inline State Settings (page 142) for a description of possible values.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNWordWrapStateTag`
: Specifies a word-wrap setting. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 81) function, you must also pass a value that specifies the line-wrapping state in the `iControlData` parameter. See Line Wrapping Settings (page 145) for a description of possible values.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNKeyboardSyncStateTag`
: Specifies whether to synchronize the keyboard with the font setting. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 81) function, you must also pass a value that specifies the keyboard synchronization state in the `iControlData` parameter. See Keyboard Synchronization Settings (page 143) for a description of possible values.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `MacTextEditor.h`.

`kTXNAutoIndentStateTag`
: Specifies an automatic indentation setting. This is available only when word warp is turned off. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 81) function, you must also pass a value that specifies the indentation state in the `iControlData` parameter. See Automatic Indentation Settings (page 113) for a description of possible values.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNTabSettingsTag`

Specifies a tab width setting. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 81) function, you must also pass a `TXNTab` (page 103) structure in the `iControlData` parameter.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNRefConTag`

An application-specific constant you define.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNMarginsTag`

Specifies margin settings. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 81) function, you must also pass a pointer to a `TXNMargins` structure in the `iControlData` parameter. You use this structure to specify the top, left, and right margin settings.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNFlattenMoviesTag`

Specifies whether to flattens movies. A flattened movie is self-contained. If you don't flatten a movie, it can't be played unless any external files (such as audio or image files) on which it depends are available. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 81) function, you must also pass a `Boolean` value in the `iControlData` parameter that specifies whether to enable (`true`) or disable (`false`) movie flattening.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDoFontSubstitution`

Specifies a font substitution setting. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 81) function, you must also pass a `Boolean` value in the `iControlData` parameter that specifies whether to enable (`true`) or disable (`false`) font substitution. For best performance, don't enable font substitution.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNNoUserIOTag`

Specifies an input setting; that is, whether to prevent input typed by the user, but allows your application to use the `TXNSetData` (page 74) function. Text objects could have read-only with respect to the application user, but have read-and-write privileges with respect to the application. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 81) function, you must also pass a value that specifies a read-write setting in the `iControlData` parameter. See Read and Write Privileges Settings (page 145) for a description of possible values.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNUseCarbonEvents`

Specifies settings for using Carbon events. (**Deprecated.** Use `TXNGetEventTarget` (page 54) and `TXNSetEventTarget` (page 75).)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDrawCaretWhenInactiveTag`

Specifies settings for drawing the caret when the text object does not have focus. (**Deprecated.** In Mac OS Xv10.4 and later, MLTE never draws the caret when the text object does not have focus.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDrawSelectionWhenInactiveTag`

Specifies settings for drawing the selection when the text object does not have focus.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDisableDragAndDropTag`

Specifies settings for drag and drop support.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNSingleLevelUndoTag`

Specifies to use a single level of undo.

Available in Mac OS X v10.2 and later.

Declared in `MacTextEditor.h`.

`kTXNVisibilityTag`

Specifies visibility.

Available in Mac OS X v10.2 and later.

Declared in `MacTextEditor.h`.

`kTXNDisableLayoutAndDrawTag`

Specifies visibility. Equivalent to `kTXNVisibilityTag`. Available in Mac OS X v10.4 and later. Use this tag to disable and re-enable layout and drawing. It optimizes performance when adding data incrementally to a text object.

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

`kTXNAutoScrollBehaviorTag`

Specifies autoscroll behavior. Available in Mac OS X v10.4 and later. For constants that represent the various types of autoscrolling, see Automatic Scrolling Behavior (page 114).

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

Discussion

You pass formatting and privileges settings in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 81) or `TXNGetTXNObjectControls` (page 59) functions. If you want to clear a setting associated with a control tag, you can set the value of the `iControlData` parameter to Clearance Settings (page 116).

Declared In
`MacTextEditor.h`

Frame Option Bits

Specify frame options for a text object.

```
enum {
    kTXNDrawGrowIconBit = 0,
    kTXNShowWindowBit = 1,
    kTXNWantHScrollBarBit = 2,
    kTXNWantVScrollBarBit = 3,
    kTXNNoTSMEverBit = 4,
    kTXNReadOnlyBit = 5,
    kTXNNoKeyboardSyncBit = 6,
    kTXNNoSelectionBit = 7,
    kTXNSaveStylesAsSTYLResourceBit = 8,
    kOutputTextInUnicodeEncodingBit = 9,
    kTXNDoNotInstallDragProcsBit = 10,
    kTXNAlwaysWrapAtViewEdgeBit = 11,
    kTXNDontDrawCaretWhenInactiveBit = 12,
    kTXNDontDrawSelectionWhenInactiveBit = 13,
    kTXNSingleLineOnlyBit = 14,
    kTXNDisableDragAndDropBit = 15,
    kTXNUseQDforImagingBit = 16,
    kTXNMonostyledTextBit = 17
};
```

Constants

`kTXNDrawGrowIconBit`

When this bit is set, it indicates the frame will have a size box. The presence of a size box in the lower right corner of an MLTE pane is only useful for resizing an MLTE pane if the MLTE pane occupies the entire window (a full-window MLTE object). In this case your application would look for a mouse-down event in the size box and call the function `TXNGrowWindow` as appropriate. Note that the size box is not supported as a means of resizing MLTE panes using `TXNGrowWindow` for MLTE pane objects.

Passing the `kTXNDrawGrowIconMask` constant to the function `TXNNewObject` only causes a size box to be drawn in the lower right corner of the MLTE pane. Passing this constant does not create a size box control in the window. The window will not contain an actual size box control. This means the window will not receive events that indicate a mouse-down event in the grow region. For this to happen, when you create the window that contains the MLTE pane, you must create the window to have a size box.

In summary, although you may pass the constants `kTXNDrawGrowIconMask` to the function `TXNNewObject` when you create an MLTE object in a window, this action only causes the visual appearance of a size box in the lower right corner of the MLTE pane. If you want to detect mouse-down events in the size box, you must also provide a size box in the window through the appropriate Window Manager functions or other tools.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNShowWindowBit`
: When this bit is set, it indicates MLTE should display a window when a text object is created. If this bit is set, your application no longer needs to call the `ShowWindow` function from the Window Manager; MLTE will do this for you.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNWantHScrollBarBit`
: When this bit is set, it indicates the frame should have a horizontal scroll bar.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNWantVScrollBarBit`
: When this bit is set, it indicates the frame should have a vertical scroll bar.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNNoTSMEverBit`
: When this bit is set, it indicates not to use Text Services Manager. You cannot use this bit when your application accepts Unicode input. (**Deprecated.** You can no longer set this because in Mac OS X, MLTE always uses the Text Services Manager.)

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNReadOnlyBit`
: When this bit is set, it indicates the text object is read-only. If you set this bit when you call the function `TXNNewObject`, the text object is put into a state that does not allow user input. However, your application can put data into the text object by calling the function `TXNSetData`. If you want the text object set into a more restrictive read-only state that does not allow user input or your application to put data into the text object programmatically you need to call the function `TXNSetTXNObjectControls`, passing the tag `kTXNIOPrivilegesTag`.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNNoKeyboardSyncBit`
: When this bit is set, it indicates no keyboard synchronization.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `MacTextEditor.h`.

`kTXNNoSelectionBit`
: When this bit is set, it indicates MLTE should not show the insertion point.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNSaveStylesAsSTYLResourceBit`
: When this bit is set, it indicates text style should be saved as a `kTXNMultipleStylesPerTextDocumentResType` resource. You can set this to assure compatibility with SimpleText. If you use `kTXNMultipleStylesPerTextDocumentResType` resources to save style info, your documents can have as many styles as you'd like. However tabs are not saved. If you don't set this bit, plain text files are saved as `kTXNSingleStylePerTextDocumentResType` resources, and only the first style in the document is saved. (Your application is expected to apply all style changes to the entire document.) If you save files with a `kTXNSingleStylePerTextDocumentResType` resource, their output is similar to those output by CodeWarrior, BBEdit, and MPW.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kOutputTextInUnicodeEncodingBit`
: When this bit is set, it indicates plain text should be saved as Unicode.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNDoNotInstallDragProcsBit`
: When this bit is set, it indicates you want to call your own drag handlers.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNAlwaysWrapAtViewEdgeBit`
: When this bit is set, it indicates line wrap at the edge of the view rectangle.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNDontDrawCaretWhenInactiveBit`
: When this bit is set, it indicates the caret should not be drawn when the object does not have focus. (**Deprecated.** In Mac OS X v10.4, MLTE never draws the caret when the text object does not have focus.)

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNDontDrawSelectionWhenInactiveBit`
: When this bit is set, it indicates the selection should not be drawn when the object does not have focus.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNSingleLineOnlyBit`
: When this bit is set, it indicates that the text object will not scroll vertically. Horizontal scrolling will stop when the end of the text is visible, and there will be no limit to the width of the text. In addition, a line break character typed, pasted, or dropped into the text object will be translated into a hyphen (-).

  Available in Mac OS X v10.1 and later.

  Declared in `MacTextEditor.h`.

`kTXNDisableDragAndDropBit`
When this bit is set, it indicates that drag and drop will not be allowed in the text object.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNUseQDforImagingBit`

When this bit is set, it indicates that QuickDraw will be used for imaging instead of Quartz, which is the default. Available in Mac OS X only. (**Deprecated.** You can no longer set the imaging system to use; MLTE always uses Quartz imaging.)

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNMonostyledTextBit`
When this bit is set, it indicates that the text object will have a single style no matter what kind of changes are made to the object.

Available in Mac OS X v10.2 and later.

Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Frame Option Masks

Represents information about frame behavior (such as whether there are scroll bars and a size box).

```
typedef OptionBits TXNFrameOptions;
enum {
    kTXNDrawGrowIconMask = 1L << kTXNDrawGrowIconBit,
    kTXNShowWindowMask = 1L << kTXNShowWindowBit,
    kTXNWantHScrollBarMask = 1L << kTXNWantHScrollBarBit,
    kTXNWantVScrollBarMask = 1L << kTXNWantVScrollBarBit,
    kTXNNoTSMEverMask = 1L << kTXNNoTSMEverBit,
    kTXNReadOnlyMask = 1L << kTXNReadOnlyBit,
    kTXNNoKeyboardSyncMask = 1L << kTXNNoKeyboardSyncBit,
    kTXNNoSelectionMask = 1L << kTXNNoSelectionBit,
    kTXNSaveStylesAsSTYLResourceMask = 1L <<
kTXNSaveStylesAsSTYLResourceBit,
    kOutputTextInUnicodeEncodingMask = 1L <<
kOutputTextInUnicodeEncodingBit,
    kTXNDoNotInstallDragProcsMask = 1L << kTXNDoNotInstallDragProcsBit,
    kTXNAlwaysWrapAtViewEdgeMask = 1L << kTXNAlwaysWrapAtViewEdgeBit,
    kTXNDontDrawCaretWhenInactiveMask = 1L <<
kTXNDontDrawCaretWhenInactiveBit,
    kTXNDontDrawSelectionWhenInactiveMask = 1L <<
kTXNDontDrawSelectionWhenInactiveBit,
    kTXNSingleLineOnlyMask = 1L << kTXNSingleLineOnlyBit,
    kTXNDisableDragAndDropMask = 1L << kTXNDisableDragAndDropBit,
    kTXNUseQDforImagingMask = 1L << kTXNUseQDforImagingBit,
    kTXNMonostyledTextMask = 1L << kTXNMonostyledTextBit
};
```

Constants

`kTXNDrawGrowIconMask`

Use to set or test for the `kTXNDrawGrowIconBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNShowWindowMask`

Use to set or test for the `kTXNShowWindowBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNWantHScrollBarMask`

Use to set or test for the `kTXNWantHScrollBarBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNWantVScrollBarMask`

Use to set or test for the `kTXNWantVScrollBarBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNNoTSMEverMask`

Use to set or test for the `kTXNNoTSMEverBit` bit. (**Deprecated.** You can no longer set this because in Mac OS X, MLTE always uses the Text Services Manager.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNReadOnlyMask`
Use to set or test for the `kTXNReadOnlyBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNNoKeyboardSyncMask`
Use to set or test for the `kTXNNoKeyboardSyncBit` bit.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNNoSelectionMask`
Use to set or test for the `kTXNNoSelectionBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNSaveStylesAsSTYLResourceMask`
Use to set or test for the `kTXNSaveStylesAsSTYLResourceBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kOutputTextInUnicodeEncodingMask`
Use to set or test for the `kOutputTextInUnicodeEncodingBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDoNotInstallDragProcsMask`
Use to set or test for the `kTXNDoNotInstallDragProcsBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNAlwaysWrapAtViewEdgeMask`
Use to set or test for the `kTXNAlwaysWrapAtViewEdgeBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDontDrawCaretWhenInactiveMask`

Use to set or test for the `kTXNDontDrawCaretWhenInactiveBit` bit. (**Deprecated.** In Mac OS Xv10.4 and later, MLTE never draws the caret when the text object does not have focus.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDontDrawSelectionWhenInactiveMask`
Use to set or test for the `kTXNDontDrawSelectionWhenInactiveBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNSingleLineOnlyMask`
Use to set or test for the `kTXNSingleLineOnlyBit` bit.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNDisableDragAndDropMask`
: Use to set or test for the `kTXNDisableDragAndDropBit` bit.

  Available in Mac OS X v10.1 and later.

  Declared in `MacTextEditor.h`.

`kTXNUseQDforImagingMask`
: Use to set or test for the `kTXNUseQDforImagingBit` bit. (**Deprecated.** You can no longer set the imaging system; MLTE always uses Quartz imaging.)

  Available in Mac OS X v10.1 and later.

  Declared in `MacTextEditor.h`.

`kTXNMonostyledTextMask`
: Use to set or test for the `kTXNMonostyledTextBit` bit.

  Available in Mac OS X v10.2 and later.

  Declared in `MacTextEditor.h`.

Discussion
See Frame Option Bits (page 133).

Declared In
`MacTextEditor.h`

HIObject Class ID

Specifies the HIObject class ID for the HITextView class.

```
#define kHITextViewClassID CFSTR("com.apple.HITextView")
```

Constants
`kHITextViewClassID`
: The class ID for the HITextView class.

Discussion
For more information on HIView, see the document *Introducing HIView*, available from the Apple Developer Documentation website.

Declared In
`MacTextEditor.h`

HIObject Control Kind

Specifies the HIObject control kind for the HITextView class.

```
enum {
    kControlKindHITextView   = 'hitx'
};
```

Constants
`kControlKindHITextView`
: The control kind for the HITextView class.

  Available in Mac OS X v10.3 and later.

  Declared in `HITextViews.h`.

Discussion

For more information on HIView, see the document *Introducing HIView*, available from the Apple Developer Documentation website.

Declared In

`MacTextEditor.h`

Initialization Option Bits

Specify initialization options for MLTE.

```
enum {
    kTXNWantMoviesBit = 0,
    kTXNWantSoundBit = 1,
    kTXNWantGraphicsBit = 2,
    kTXNAlwaysUseQuickDrawTextBit = 3,
    kTXNUseTemporaryMemoryBit = 4
};
```

Constants

`kTXNWantMoviesBit`

When this bit is set, it specifies that movie data is supported in a text object.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNWantSoundBit`

When this bit is set, it specifies that sound data is supported in a text object.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNWantGraphicsBit`

When this bit is set, it specifies that graphics data is supported in a text object.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNAlwaysUseQuickDrawTextBit`

When this bit is set, it specifies that MLTE should use QuickDraw for imaging even if ATSUI is available. This is often the best choice for applications that need speed and efficient memory usage.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNUseTemporaryMemoryBit`

When this bit is set, it specifies that MLTE should use temporary memory for all memory allocations.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Initialization Option Masks

Represents MLTE initialization options.

```
typedef OptionBits TXNInitOptions;
enum {
    kTXNWantMoviesMask = 1L << kTXNWantMoviesBit,
    kTXNWantSoundMask = 1L << kTXNWantSoundBit,
    kTXNWantGraphicsMask = 1L << kTXNWantGraphicsBit,
    kTXNAlwaysUseQuickDrawTextMask = 1L << kTXNAlwaysUseQuickDrawTextBit,
    kTXNUseTemporaryMemoryMask = 1L << kTXNUseTemporaryMemoryBit
};
```

Constants

`kTXNWantMoviesMask`

Use to set or test for the `kTXNWantMoviesBit` bit.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNWantSoundMask`

Use to set or test for the `kTXNWantSoundBit` bit.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNWantGraphicsMask`

Use to set or test for the `kTXNWantGraphicsBit` bit.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNAlwaysUseQuickDrawTextMask`

Use to set or test for the `kTXNAlwaysUseQuickDrawTextBit` bit.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNUseTemporaryMemoryMask`

Use to set or test for the `kTXNUseTemporaryMemoryBit` bit.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

Discussion
Used in the `iUsageFlags` parameter of the `TXNInitTextension` (page 62) function.

Declared In
`MacTextEditor.h`

Inline State Settings

Specify the inline state for a text object.

```
enum {
    kTXNUseInline = false,
    kTXNUseBottomline = true
};
```

Constants

`kTXNUseInline`

Text is entered at the caret insertion point in the text object's window.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNUseBottomline`

Text is entered in a bottom-line window (a small floating window that appears at the bottom of the screen).

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion

Passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 81) function when the value of the `iControlTags` parameter is `kTXNInlineStateTag`, or returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 59) function when the value of the `iControlTags` parameter is `kTXNInlineStateTag`.

Declared In

`MacTextEditor.h`

Justification Settings

Specify the justification setting.

```
enum {
    kTXNFlushDefault = 0,
    kTXNFlushLeft = 1,
    kTXNFlushRight = 2,
    kTXNCenter = 4,
    kTXNFullJust = 8,
    kTXNForceFullJust = 16
};
```

Constants

`kTXNFlushDefault`

Justification is flush according to the line direction.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNFlushLeft`

Justification is flush left.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNFlushRight`
: Justification is flush right.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNCenter`
: Justification is centered.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNFullJust`
: Justification is flush left and right for all lines except the last line in a paragraph.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNForceFullJust`
: Justification is flush left and right for all lines.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

Discussion

Passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 81) function when the value of the `iControlTags` parameter is `kTXNJustificationTag`, or returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 59) function when the value of the `iControlTags` parameter is `kTXNJustificationTag`.

Declared In

`MacTextEditor.h`

Keyboard Synchronization Settings

Specify the keyboard synchronization setting for a text object.

```
enum {
    kTXNSyncKeyboard = false,
    kTXNNoSyncKeyboard = true
};
```

Constants

`kTXNSyncKeyboard`
: Keyboard synchronization is enabled.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `MacTextEditor.h`.

`kTXNNoSyncKeyboard`
: Keyboard synchronization is not enabled.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `MacTextEditor.h`.

Discussion
These constants are passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 81) function when the value of the `iControlTags` parameter is `kTXNKeyboardSyncStateTag`. They are also returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 59) function when the value of the `iControlTags` parameter is `kTXNKeyboardSyncStateTag`.

Declared In
`MacTextEditor.h`

Layout and Draw Constants

Specify the layout and draw setting for a text object.

```
enum {
    kTXNEnableLayoutAndDraw = false,
    kTXNDisableLayoutAndDraw = true
};
```

Constants
`kTXNEnableLayoutAndDraw`
Layout and draw is enabled.

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

`kTXNDisableLayoutAndDraw`
Layout and draw is disabled.

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

Discussion
These constants are passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 81) function when the value of the `iControlTags` parameter is `kTXNDisableLayoutAndDrawTag`. They are also are returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 59) function when the value of the `iControlTags` parameter is `kTXNDisableLayoutAndDrawTag`.

Declared In
`MacTextEditor.h`

Line Direction Settings

Specify the line direction setting.

```
enum {
    kTXNLeftToRight = 0,
    kTXNRightToLeft = 1
};
```

Constants
`kTXNLeftToRight`
Line direction flows from left to right.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNRightToLeft`
: Line direction flows from right to left.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

Discussion

Passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 81) function when the value of the `iControlTags` parameter is `kTXNLineDirectionTag`, or returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 59) function when the value of the `iControlTags` parameter is `kTXNLineDirectionTag`.

Declared In

`MacTextEditor.h`

Line Wrapping Settings

Specify the line-wrap setting for a text object.

```
enum {
    kTXNAutoWrap = false,
    kTXNNoAutoWrap = true
};
```

Constants

`kTXNAutoWrap`
: Automatic line wrapping is enabled.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNNoAutoWrap`
: Automatic line wrapping is not enabled.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

Discussion

Passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 81) function when the value of the `iControlTags` parameter is `kTXNWordWrapStateTag`, or returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 59) function when the value of the `iControlTags` parameter is `kTXNWordWrapStateTag`.

Declared In

`MacTextEditor.h`

Read and Write Privileges Settings

Specify the privileges setting for a text object.

```
enum {
    kTXNReadWrite = false,
    kTXNReadOnly = true
};
```

Constants

`kTXNReadWrite`
: The document has read and write privileges.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNReadOnly`
: The document is read-only.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

Discussion

Passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 81) function when the value of the `iControlTags` parameter is `kTXNIOPrivilegesTag`, or returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 59) function when the value of the `iControlTags` parameter is `kTXNIOPrivilegesTag`.

Declared In

`MacTextEditor.h`

Rectangle Keys

Specifies the bounds to use for a text object.

```
typedef UInt32    TXNRectKey;
enum {
    kTXNViewRectKey                = 0,
    kTXNDestinationRectKey         = 1,
    kTXNTextRectKey                = 2,
    kTXNVerticalScrollBarRectKey   = 3,
    kTXNHorizontalScrollBarRectKey = 4
};
```

Constants

`kTXNViewRectKey`
: Specifies to use the view rectangle.

  Available in Mac OS X v10.3 and later.

  Declared in `MacTextEditor.h`.

`kTXNDestinationRectKey`
: Specifies to use the destination rectangle.

  Available in Mac OS X v10.3 and later.

  Declared in `MacTextEditor.h`.

`kTXNTextRectKey`
: Specifies to use the text rectangle.

  Available in Mac OS X v10.3 and later.

  Declared in `MacTextEditor.h`.

`kTXNVerticalScrollBarRectKey`
: Specifies to include the vertical scroll bar in the rectangle.

  Available in Mac OS X v10.3 and later.

  Declared in `MacTextEditor.h`.

`kTXNHorizontalScrollBarRectKey`
: Specifies to include the horizontal scroll bar in the rectangle.

  Available in Mac OS X v10.3 and later.

  Declared in `MacTextEditor.h`.

Discussion
You can pass a rectangle key to the function `TXNGetHIRect` (page 54).

Declared In
`MacTextEditor.h`

Scroll Bar Orientation

Specifies the orientation of a text window’s scroll bar.

```
typedef UInt32 TXNScrollBarOrientation;
enum {
    kTXNHorizontal = 0,
    kTXNVertical = 1
};
```

Constants

`kTXNHorizontal`
: Specifies a horizontal scroll bar.

  Available in Mac OS X v10.2 and later.

  Declared in `MacTextEditor.h`.

`kTXNVertical`
: Specifies a vertical scroll bar.

  Available in Mac OS X v10.2 and later.

  Declared in `MacTextEditor.h`.

Discussion
You use these constants in your `TXNScrollInfoProcPtr` (page 92) callback function.

Declared In
`MacTextEditor.h`

Scroll Bar States

Represents the scroll bar state for the window attached to a text object.

```
typedef Boolean TXNScrollBarState;
enum {
    kScrollBarsAlwaysActive = true,
    kScrollBarsSyncWithFocus = false
};
```

Constants

`kScrollBarsAlwaysActive`

Indicates that scroll bars should always appear in the active state even then the text area does not have focus.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kScrollBarsSyncWithFocus`

Indicates that scroll bars should be active only if the frame in which they are enclosed is focused.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion

Used in the `TXNActivate` (page 164) function.

Declared In

`MacTextEditor.h`

Scroll Units

Specify the unit by which scrolling should occur.

```
typedef UInt32 TXNScrollUnit;
enum {
    kTXNScrollUnitsInPixels = 0,
    kTXNScrollUnitsInLines = 1,
    kTXNScrollUnitsInViewRects = 2
};
```

Constants

`kTXNScrollUnitsInPixels`

Specifies pixels as the scrolling unit.

Available in Mac OS X v10.2 and later.

Declared in `MacTextEditor.h`.

`kTXNScrollUnitsInLines`

Specifies line count as the scrolling unit. Scrolling is slower when you use this unit because each line must be measured by MLTE before the text scrolls.

Available in Mac OS X v10.2 and later.

Declared in `MacTextEditor.h`.

`kTXNScrollUnitsInViewRects`

Specifies the height of the current view rectangle as the scrolling unit.

Available in Mac OS X v10.2 and later.

Declared in `MacTextEditor.h`.

Discussion
These constants are supplied as the `iVerticalScrollUnit` and `iHorizontalScrollUnit` parameters to the `TXNScroll` (page 70) function. They specifies the units used for each of these parameters.

Declared In
`MacTextEditor.h`

Search Criteria Bits

Specify matching rules to use when searching.

```
enum {
    kTXNIgnoreCaseBit = 0,
    kTXNEntireWordBit = 1,
    kTXNUseEncodingWordRulesBit = 31
};
```

Constants

`kTXNIgnoreCaseBit`
When this bit is set, indicates that case should be ignored.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNEntireWordBit`
When this bit is set, indicates that the entire word must match.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNUseEncodingWordRulesBit`
When this bit is set, indicates that Unicode Utilities should be used to find a word boundary. You need to set this bit if your applications uses 2-byte scripts in which words are not separated by a space.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Search Criteria Masks

Represents the matching rules to be used in a find operation.

```
typedef OptionBits TXNMatchOptions;
enum {
    kTXNIgnoreCaseMask = 1L << kTXNIgnoreCaseBit,
    kTXNEntireWordMask = 1L << kTXNEntireWordBit,
    kTXNUseEncodingWordRulesMask = 1L << kTXNUseEncodingWordRulesBit
};
```

Constants

`kTXNIgnoreCaseMask`

Use to set or test for the `kTXNIgnoreCaseBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNEntireWordMask`

Use to set or test for the `kTXNEntireWordBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNUseEncodingWordRulesMask`

Use to set or test for the `kTXNEntireWordBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion

These constants can be passed in the `iMatchOptions` parameter of the `TXNFind` (page 45) function or the callback `TXNFindProcPtr` (page 90).

Declared In

`MacTextEditor.h`

Selection Display Settings

Specify whether the text object should scroll to show the beginning or the end of the selection.

```
enum {
    kTXNShowStart = false,
    kTXNShowEnd = true
};
```

Constants

`kTXNShowStart`

The start of the selection should be shown. The selection scrolls to show the start if necessary.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNShowEnd`

The end of the selection should be shown. The selection scrolls to show the end if necessary.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Declared In

`MacTextEditor.h`

Selection State Settings

Specify whether or not MLTE displays a caret and allows selections in text that is read-only.

```
enum {
    kTXNSelectionOn = true,
    kTXNSelectionOff = false
};
```

Constants

`kTXNSelectionOn`

Display a caret and allow a selection in text that is read-only.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNSelectionOff`

Do not display a caret or allow a selection in text that is read-only.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion

Passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 81) function when the value of the `iControlTags` parameter is `kTXNSelectionStateTag`, or returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 59) function when the value of the `iControlTags` parameter is `kTXNSelectionStateTag`.

Declared In

`MacTextEditor.h`

Style Resource Types

Specify the resource type to use to save style information for a plain text document.

```
enum {
    kTXNSingleStylePerTextDocumentResType = 'MPSR',
    kTXNMultipleStylesPerTextDocumentResType = 'styl'
};
```

Constants

`kTXNSingleStylePerTextDocumentResType`

User for a document that contains a single style and should be treated as a BBEdit, MPW, or CodeWarrior document.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNMultipleStylesPerTextDocumentResType`

Use for a document that contains multiple styles and should be treated as a SimpleText document.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion

Used in the `TXNNewObject` (page 170) function.

Declared In
`MacTextEditor.h`

Supported Data Types

Specifies the type of data being requested from or passed to an MLTE function.

```
typedef OSType TXNDataType;
enum {
    kTXNTextData = 'TEXT',
    kTXNPictureData = 'PICT',
    kTXNMovieData = 'moov',
    kTXNSoundData = 'snd ',
    kTXNUnicodeTextData = 'utxt'
};
```

Constants

`kTXNTextData`

Text data.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNPictureData`

Graphics (`PICT`) data.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNMovieData`

Movie or sound data.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNSoundData`

Sound data.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNUnicodeTextData`

Unicode text data.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion
Used in the `TXNGetDataEncoded` (page 53) function.

Declared In
`MacTextEditor.h`

Supported File Types

Represents a file type.

```
typedef OSType TXNFileType;
enum {
    kTXNTextensionFile = 'txtn',
    kTXNTextFile = 'TEXT',
    kTXNPictureFile = 'PICT',
    kTXNMovieFile = 'MooV',
    kTXNSoundFile = 'sfil',
    kTXNAIFFFile = 'AIFF',
    kTXNUnicodeTextFile = 'utxt'
};
```

Constants

`kTXNTextensionFile`

A file that contains Unicode or Mac OS text. By default, it contains Unicode text. Files are saved in a private format.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNTextFile`

A file that contains plain text data.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNPictureFile`

A file that contains graphics data in `PICT` format.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNMovieFile`

A file that contains movie data in `'MooV'` format.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNSoundFile`

A file that contains sound data in `'sfil'` format.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNAIFFFile`

A file that contains sound data in `'aiff'` format.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNUnicodeTextFile`
: A file that contains Unicode text data.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Supported Frame Types

Represents a frame type. (**Deprecated.** No longer needed.)

```
typedef UInt32 TXNFrameType;
enum {
    kTXNTextEditStyleFrameType = 1,
    kTXNPageFrameType = 2,
    kTXNMultipleFrameType = 3
};
```

Constants
`kTXNTextEditStyleFrameType`
: A single rectangle that allows text to scroll if the rectangle fills. Although you can pass this as a parameter to the function `TXNNewObject` (page 170), you should instead use the function `TXNCreateObject` (page 35), which does not require a frame type.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNPageFrameType`
: A single rectangle with a bottom. That is, text moves to a new page if the frame is full. This constant is not supported in Mac OS X version 10.3 and later.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNMultipleFrameType`
: Multiple frames. This constant is not supported in Mac OS X version 10.3 and later.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

Special Considerations

This data type is used only by the `TXNNewObject` function, which is deprecated.

Declared In
`MacTextEditor.h`

Tab Types

Defines the tab settings for a text object.

```
typedef SInt8 TXNTabType;
enum {
    kTXNRightTab = -1,
    kTXNLeftTab = 0,
    kTXNCenterTab = 1
};
```

Constants

`kTXNRightTab`

Right tabs are active.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNLeftTab`

Left tabs are active; not available in MLTE version 1.0.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNCenterTab`

Center tabs are active; not available in MLTE version 1.0.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion

Used in the `TXNTab` (page 103) structure.

Declared In

`MacTextEditor.h`

Text Background Types

Represents a background data type used in the `TXNBackground` structure.

```
typedef UInt32 TXNBackgroundType;
enum {
    kTXNBackgroundTypeRGB = 1
};
```

Constants

`kTXNBackgroundTypeRGB`

Indicates color.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion

Used in the `TXNBackground` (page 95) structure. MLTE supports only color as the background type.

Declared In

`MacTextEditor.h`

Text Box Options Bits

Specify how text should be displayed in a Unicode text box.

```
enum {
    kTXNSetFlushnessBit = 0,
    kTXNSetJustificationBit = 1,
    kTXNUseFontFallBackBit = 2,
    kTXNRotateTextBit = 3,
    kTXNUseVerticalTextBit = 4,
    kTXNDontUpdateBoxRectBit = 5,
    kTXNDontDrawTextBit = 6,
    kTXNUseCGContextRefBit = 7,
    kTXNImageWithQDBit = 8,
    kTXNDontWrapTextBit = 9
};
```

Constants

`kTXNSetFlushnessBit`

When this bit is set, indicates MLTE should display text flush according to the line direction.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNSetJustificationBit`

When this bit is set, indicates justification. Text is justified in the direction of the display. Horizontal text is justified horizontally, but not vertically. Vertical text is justified vertically, but not horizontally.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNUseFontFallBackBit`

When this bit is set, indicates MLTE should use ATSUI transient font matching that searches for a font that has a matching character.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNRotateTextBit`

When this bit is set, indicates MLTE should display text rotated clockwise.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNUseVerticalTextBit`

When this bit is set, indicates MLTE should display text vertically from top to bottom.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNDontUpdateBoxRectBit`

When this bit is set, indicates MLTE should not update the specified rectangle. If you set this bit when you call the `TXNDrawUnicodeTextBox` (page 42) function, the function does not update the right coordinates of the specified rectangle to accommodate the longest line for text.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNDontDrawTextBit`

When this bit is set, indicates MLTE should return the size of the text but should not draw the text box.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNUseCGContextRefBit`

When this bit is set, indicates MLTE should the Quartz context (`CGContext`) you provide instead of the temporary `CGContextRef` created internally by MLTE. To do so, you must set the `kTXNUseCGContextRefBit` bit in `TXNTextBoxOptions` and pass a `CGContextRef` in the `options` field of the `TXNTextBoxOptionsData` structure.

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNImageWithQDBit`

When this bit is set, indicates MLTE should use QuickDraw for imaging text. (**Deprecated.** You can no longer set the imaging system; MLTE always uses Quartz imaging.)

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNDontWrapTextBit`

When this bit is set, indicates MLTE should not wrap text.

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Text Box Options Masks

Defines how text appears in a text box.

```
typedef OptionBits TXNTextBoxOptions;
enum {
    kTXNSetFlushnessMask = 1L << kTXNSetFlushnessBit,
    kTXNSetJustificationMask = 1L << kTXNSetJustificationBit,
    kTXNUseFontFallBackMask = 1L << kTXNUseFontFallBackBit,
    kTXNRotateTextMask = 1L << kTXNRotateTextBit,
    kTXNUseVerticalTextMask = 1L << kTXNUseVerticalTextBit,
    kTXNDontUpdateBoxRectMask = 1L << kTXNDontUpdateBoxRectBit,
    kTXNDontDrawTextMask = 1L << kTXNDontDrawTextBit,
    kTXNUseCGContextRefMask = 1L << kTXNUseCGContextRefBit,
    kTXNImageWithQDMask = 1L << kTXNImageWithQDBit,
    kTXNDontWrapTextMask = 1L << kTXNDontWrapTextBit
};
```

Constants

`kTXNSetFlushnessMask`

Use to set or test for the `kTXNSetFlushnessBit` bit.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNSetJustificationMask`

Use to set or test for the `kTXNSetJustificationBit` bit.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNUseFontFallBackMask`

Use to set or test for the `kTXNUseFontFallBackBit` bit.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNRotateTextMask`

Use to set or test for the `kTXNRotateTextBit` bit.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNUseVerticalTextMask`

Use to set or test for the `kTXNUseVerticalTextBit` bit.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNDontUpdateBoxRectMask`

Use to set or test for the `kTXNDontUpdateBoxRectBit` bit.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNDontDrawTextMask`
: Use to set or test for the `kTXNDontDrawTextBit` bit.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `MacTextEditor.h`.

`kTXNUseCGContextRefMask`
: Use to set or test for `kTXNUseCGContextRefBit` bit.

  Available in Mac OS X v10.1 and later.

  Not available to 64-bit applications.

  Declared in `MacTextEditor.h`.

`kTXNImageWithQDMask`
: Use to set or test for `kTXNImageWithQDBit` bit. (**Deprecated.** You can no longer set the imaging system; MLTE always uses Quartz imaging.)

  Available in Mac OS X v10.1 and later.

  Declared in `MacTextEditor.h`.

`kTXNDontWrapTextMask`
: Use to set or test for `kTXNDontWrapTextBit` bit.

  Available in Mac OS X v10.1 and later.

  Not available to 64-bit applications.

  Declared in `MacTextEditor.h`.

Discussion
Used in the `TXNDrawUnicodeTextBox` (page 42) and `TXNDrawCFStringTextBox` (page 41) functions.

Declared In
`MacTextEditor.h`

Text Encoding Preferences

Represents how to encode text for your application.

```
typedef UInt32 TXNPermanentTextEncodingType;
enum {
    kTXNSystemDefaultEncoding = 0,
    kTXNMacOSEncoding = 1,
    kTXNUnicodeEncoding = 2
};
```

Constants

`kTXNSystemDefaultEncoding`
: Use the encoding that is used internally by MLTE and the system. The preferred encoding is Unicode for a system that has ATSUI.

  Available in Mac OS X v10.0 and later.

  Declared in `MacTextEditor.h`.

`kTXNMacOSEncoding`
Incoming and outgoing text should be in traditional Mac OS script system encodings even if MLTE uses another format internally. MLTE will use the Text Encoding Convertor (TEC) to convert text and offsets to match your application's preference.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNUnicodeEncoding`
Incoming and outgoing text should be in Unicode even on systems that do not have ATSUI. MLTE will use the Text Encoding Convertor (TEC) to convert text and offsets to match the applications preference.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion
These convenience constants can be used in the functions `TXNNewObject` (page 170) and `TXNSave` (page 180).

Declared In
`MacTextEditor.h`

Result Codes

The most common result codes returned by MLTE are listed below.

Result Code	Value	Description
`kTXNEndIterationErr`	-22000	Function was not able to iterate through the data contained by a text object. Available in Mac OS X v10.0 and later.
`kTXNCannotAddFrameErr`	-22001	Frame was not added. The multiple-frame feature is currently not available in MLTE, so this error is returned any time you try to define an object with multiple frames. Available in Mac OS X v10.0 and later.
`kTXNInvalidFrameIDErr`	-22002	The frame ID is invalid. Available in Mac OS X v10.0 and later.
`kTXNIllegalToCrossDataBoundariesErr`	-22003	Offsets specify a range that crosses a data type boundary. Available in Mac OS X v10.0 and later.
`kTXNUserCanceledOperationErr`	-22004	A user canceled an operation before your application completed processing it. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`kTXNBadDefaultFileTypeWarning`	-22005	Text file is not in the format you specified. Available in Mac OS X v10.0 and later.
`kTXNCannotSetAutoIndentErr`	-22006	Automatic indenting could not be enabled—the document has word wrapping enabled and you tried to enable auto indentation. Available in Mac OS X v10.0 and later.
`kTXNRunIndexOutofBoundsErr`	-22007	An index you supplied to a function is out of bounds. Available in Mac OS X v10.0 and later.
`kTXNNoMatchErr`	-22008	Returned by `TXNFind` (page 45) when a match is not found. Available in Mac OS X v10.0 and later.
`kTXNAttributeTagInvalidForRunErr`	-22009	Tag for a specific run is not valid. Available in Mac OS X v10.0 and later.
`kTXNSomeOrAllTagsInvalidForRunErr`	-22010	Tags supplied to a function are not valid. Available in Mac OS X v10.0 and later.
`kTXNInvalidRunIndex`	-22011	Index is out of range for that run. Available in Mac OS X v10.0 and later.
`kTXNAlreadyInitializedErr`	-22012	You already called the `TXNInitTextension` (page 62) function. Available in Mac OS X v10.0 and later.
`kTXNCannotTurnTSMOffWhenUsingUnicodeErr`	-22013	Your application tried to turn off the Text Services Manager while MLTE was set to process Unicode. Available in Mac OS X v10.0 and later.
`kTXNCopyNotAllowedInEchoModeErr`	-22014	Your application tried to copy text that was in echo mode. Available in Mac OS X v10.0 and later.
`kTXNDataTypeNotAllowedErr`	-22015	Your applications specifies a data type that MTLE does not allow. Available in Mac OS X v10.0 and later.
`kTXNATSUIIsNotInstalledErr`	-22016	Indicates ATSUI is not installed on the system. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`kTXNOutsideOfLineErr`	-22017	Indicates a value that is beyond the length of the line. Available in Mac OS X v10.0 and later.
`kTXNOutsideOfFrameErr`	-22018	Indicates a value that is outside of the text object's frame. Available in Mac OS X v10.0 and later.
`kTXNDisabledFunctionalityErr`	-22019	Indicates the function has been disabled. Available in Mac OS X v10.3 and later.
`kTXNOperationNotAllowedErr`	-22020	Indicates that the function cannot be called in this context. Available in Mac OS X v10.4 and later.

Deprecated Multilingual Text Engine Functions

A function identified as deprecated has been superseded and may become unsupported in the future.

Deprecated in Mac OS X v10.2

TXNSetViewRect

Sets the rectangle that describes the current view into the document; changes the amount of text that is viewable. (Deprecated in Mac OS X v10.2. Use `TXNSetFrameBounds` (page 77) or `TXNSetRectBounds` (page 174) instead.)

Not recommended.

```
void TXNSetViewRect (
   TXNObject iTXNObject,
   const Rect *iViewRect
);
```

Parameters

`iTXNObject`
The text object for the current text area.

`iViewRect`
On input, points to a rectangle that describes the new view of the document.

Discussion
The `TXNSetViewRect` function does not change where a line of text wraps. Line wrapping is controlled by the `TXNSetFrameBounds` (page 77) function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.2.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

Deprecated in Mac OS X v10.3

TXNActivate

Sets the state of the scroll bars so they are drawn correctly in response to activate events. (Deprecated in Mac OS X v10.3. Use `TXNSetScrollbarState` (page 79) instead.)

Not recommended.

```
OSStatus TXNActivate (
   TXNObject iTXNObject,
   TXNFrameID iTXNFrameID,
   TXNScrollBarState iActiveState
);
```

Parameters

`iTXNObject`
The text object that identifies the document to be activated.

`iTXNFrameID`
The frame ID of the document that is to be activated. You obtain a frame ID from `TXNNewObject` (page 170) when you create a text object.

`iActiveState`
A value that indicates the state of the scroll bars. See Scroll Bar States (page 147) for a description of possible values. If you pass the `kScrollBarsAlwaysActive` constant, the scroll bars are always active, whether or not the frame's text area currently has keyboard focus. Passing `kScrollBarsAlwaysActive` can be useful for a window such as a dialog box that may contain multiple text areas, each of which may have a scrollable frame. If you pass `kScrollBarsSyncWithFocus`, MLTE synchronizes the activity state of the scroll bars with the focus state of the frame. Therefore, only when the frame has keyboard focus does it have active scroll bars. A value of `kScrollBarsSyncWithFocus` is the default and is typically recommended if you have only one frame per window.

Return Value

A result code. See "MLTE Result Codes" (page 160). `TXNActivate` returns a parameter error if you pass an invalid text object or frame ID.

Discussion

You typically call `TXNActivate` in response to an activate event. If the text object was previously inactive, `TXNActivate` removes any visual indication of its prior inactive state (such as a dimmed or framed selection area or inactive scroll bars). Before you call the `TXNActivate` function, you should make sure that the window belongs to your application.

The `TXNActivate` function does not change the keyboard focus. This means your application can have a text area that is not focused, but in which the scroll bars are active. This lets application users scroll the inactive text without changing the focus from another text area.

If you want to display a text area that has both keyboard focus and active scroll bars, you must call the `TXNFocus` (page 47) function immediately before you call the `TXNActivate` function. Note that MLTE does not retain information about keyboard focus. So if, for example, you set the keyboard focus on a text area and the window containing the text area becomes deactivated, you must call the `TXNFocus` (page 47) function when the window becomes activated again.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNAttachObjectToWindow

Attaches a text object to a window. (Deprecated in Mac OS X v10.3. Use `TXNAttachObjectToWindowRef` (page 29) instead.)

Not recommended.

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

Parameters

`iTXNObject`
The text object with which you want to associate the window.

`iWindow`
A pointer to the graphics port to which the object should be attached. The graphics port may be a window (`WindowRef`) or a generic graphics port (`CGrafPtr`, `GWorldPtr`). If it is a window, note that you must typecast the window reference to a `GWorldPtr` data type.

`iIsActualWindow`
A `Boolean` value. Pass `true` if the `iWindow` parameter you passed refers to a Window Manager window (`WindowRef`), not a generic graphics port. Pass `false` if the `iWindow` parameter you passed does not refer to a window. If you pass `false`, MLTE never calls window-specific functions such as `InvalRect` or `BeginUpdate` for this text object, and it is your program's responsibility to handle any window-related functionality.

Return Value
A result code. See "MLTE Result Codes" (page 160). `TXNAttachObjectToWindow` returns `paramErr` if the text object that you pass is invalid.

Discussion
You may create a text object without an associated window pointer by passing `NULL` in the `iWindow` parameter of the `TXNNewObject` (page 170) function. However, if you do so, you must call the `TXNAttachObjectToWindow` function to associate a window with that object before you call any other MLTE function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNConvertFromPublicScrap

Converts the Clipboard content to the private MLTE scrap. (Deprecated in Mac OS X v10.3. This function isn't needed in Mac OS X.)

Not recommended.

```
OSStatus TXNConvertFromPublicScrap (
   void
);
```

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
You should call the `TXNConvertFromPublicScrap` function after another application has modified the contents of the Clipboard. Calling the `TXNConvertFromPublicScrap` function ensures that the contents of the system Clipboard are available to your application. Typically, when you receive a resume event, you call the Scrap Manager function `GetCurrentScrap` to determine whether the Clipboard content has been modified. If so, you should then call `TXNConvertFromPublicScrap`.

Special Considerations

The function `TXNConvertFromPublicScrap` is no longer needed in Mac OS X version 10.2 and later. Calling the function `TXNPaste` automatically handles conversion from public scrap.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNConvertToPublicScrap

Converts the private MLTE scrap content to the Clipboard. (Deprecated in Mac OS X v10.3. This function isn't needed in Mac OS X.)

Not recommended.

```
OSStatus TXNConvertToPublicScrap (
   void
);
```

Return Value
A result code. See "MLTE Result Codes" (page 160).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNDraw

Redraws the text area, including any scroll bars associated with the text frame. (Deprecated in Mac OS X v10.3. Use the `TXNDrawObject` (page 42).)

Not recommended.

```
void TXNDraw (
   TXNObject iTXNObject,
   GWorldPtr iDrawPort
);
```

Parameters

`iTXNObject`

The text object whose text is to be redrawn.

`iDrawPort`

A value of type `GWorldPtr`. Pass a valid pointer or `NULL`. If you pass `NULL`, the `TXNDraw` function redraws the text area into the port that is currently associated with the text object. If you pass a valid pointer instead of `NULL`, `TXNDraw` redraws the text area into the specified port, and does not update the selection. You should pass `NULL` if you want to draw on the screen but pass a valid pointer if you want to take a snapshot of the screen to save or print.

Discussion

You can call the `TXNDraw` function in response to an update event for a window that contains multiple text objects or other graphic elements. If necessary, your application is also responsible for calling the Window Manager functions `BeginUpdate` and `EndUpdate` in response to the update event.

If there is nothing in your window except a single MLTE text object, you should call the `TXNUpdate` (page 85) function to redraw the area instead of calling `TXNDraw`. The `TXNUpdate` f unction draws everything in the frame, and you do not have to call the Window Manager functions `BeginUpdate` and `EndUpdate` yourself.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNGetRectBounds

Obtains the values for the current view, destination, and text rectangles. (Deprecated in Mac OS X v10.3. Use `TXNGetHIRect` (page 54) instead.)

Not recommended.

```
OSStatus TXNGetRectBounds (
   TXNObject iTXNObject,
   Rect *oViewRect,
   TXNLongRect *oDestinationRect,
   TXNLongRect *oTextRect
);
```

Parameters

iTXNObject

The text object for the current text area.

oViewRect

On output, a pointer to the `Rect` data structure that contains the coordinates for the view rectangle. If you do not want to obtain this structure, pass `NULL`. The view rectangle specifies the area of the text you see. Scroll bars are drawn inside the view rectangle.

oDestinationRect

On output, a pointer to the `TXNLongRect` data structure that contains the coordinates for the destination rectangle. If you do not want to obtain this structure, pass `NULL`. The destination rectangle controls how text is laid out.

oTextRect

On output, a pointer to the `TXNLongRect` data structure that contains the coordinates for the text rectangle. If you do not want to obtain this structure, pass `NULL`. The text rectangle is the smallest rectangle needed to contain the current text. MLTE calculates the text rectangle by measuring each line of text. So this can be slow performance. The width of the text rectangle is the width of the longest line in the text.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

You need only to pass pointers for the rectangles for which you want to obtain coordinates.

Availability

Available in Mac OS X v10.1 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNIsObjectAttachedToSpecificWindow

Determines whether a text object is attached to a specified window. (Deprecated in Mac OS X v10.3. Use `TXNGetWindowRef` (page 60) instead.)

Not recommended.

```
OSStatus TXNIsObjectAttachedToSpecificWindow (
   TXNObject iTXNObject,
   WindowRef iWindow,
   Boolean *oAttached
);
```

Parameters

iTXNObject

The text object that identifies the document you want to check.

iWindow

A reference to the window against which to check attachment.

oAttached

On output, `true` if the text object is attached to the specified window; `false` otherwise.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNIsObjectAttachedToWindow

Checks to see if a text object is attached to a window. (Deprecated in Mac OS X v10.3. Use `TXNGetWindowRef` (page 60) instead.)

Not recommended.

```
Boolean TXNIsObjectAttachedToWindow (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object that identifies the document you want to check.

Return Value

A `Boolean` value; returns `true` if the object is attached.

Discussion

You can call this before you call the `TXNAttachObjectToWindow` (page 165) function to make sure the text object is not already attached to a window. If you pass `NULL` in the `iWindow` parameter of `TXNNewObject` (page 170) you create a text object without an associated window pointer.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNNewObject

Creates a new MLTE text object which is an opaque structure that handles text formatting at the document level. (Deprecated in Mac OS X v10.3. Use `TXNCreateObject` (page 35) instead.)

Not recommended.

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

Parameters

`iFileSpec`

A pointer to a variable of type `FSSpec`. If you pass `NULL` you start with an empty document. Otherwise, the contents of the file to which `iFileSpec` points are read into the object. The referenced file must consist entirely of data that MLTE can read (`'TEXT'`, `'RTF '`, `'utxt'`, or `'txtn'`). If the referenced file contains your application's private data and data that MLTE can read, you should call the `TXNNewObject` function with the `iFileSpec` parameter set to `NULL`. Once `TXNNewObject` creates the text object, your application can read the private data into the text object by calling the `TXNSetDataFromFile` (page 173) function.

`iWindow`

A reference to the window in which the document will be displayed. This parameter can be `NULL`. If it is `NULL`, you must attach a window or graphics port to the text object by using the `TXNAttachObjectToWindow` (page 165) function.

`iFrame`

A pointer to a variable of type `Rect`. If you pass `NULL`, the window's `portRect` rectangle is used as the frame. If you do not want to fill the entire window, you use the `iFrame` parameter to specify the area to fill.

`iFrameOptions`

A value that specifies the options you want the frame to support. See Frame Option Masks (page 136) for a description of possible values.

If you want to create a read-only document, you need to pass the option `kTXNReadOnlyMask`. Note that this option puts the text object into a state that does not allow user input. However, your application can put data into the text object by calling the function `TXNSetData`. If you want the text object set into a more restrictive read-only state that does not allow user input or your application to put data into the text object programmatically, you need to call the function `TXNSetTXNObjectControls`, passing the tag `kTXNIOPrivilegesTag`. If you choose to set the text object into this restrictive state, you will get an error if you try to call the function `TXNSetData` on the text object. (In this case, you can change the text object to a less restrictive state by calling `TXNSetTXNObjectControls`, passing the tag `kTXNNoUserIOTag`.)

iFrameType
A value that specifies the frame type of the text object. See Supported Frame Types (page 154) for a description of possible values.

iFileType
A value that specifies the file type of the text object. See Supported File Types (page 153) for a description of possible values. You should specify the primary file type. If you use the `kTXNTextensionFile` constant, files are saved in a custom format. If you want saved files to be plain text files, you should specify the `kTXNTextFile` constant, then use the `iframeOptions` parameter to specify whether the plain text files should be saved with `kTXNSingleStylePerTextDocumentResType` or `kTXNMultipleStylesPerTextDocumentResType` resources.

iPermanentEncoding
A value that specifies the encoding in which the document is saved. See Text Encoding Preferences (page 159) for a description of possible values.

oTXNObject
A pointer to a structure of type `TXNObject`. On return, this points to the opaque text object data structure allocated by the function. You need to pass this object to most MLTE functions.

oTXNFrameID
On return, a pointer to the unique ID for the text object's frame. However, in MLTE version 1.1 and earlier, the frame ID is always set to 0.

iRefCon
A value of type `TXNObjectRefcon`. You can define how to use this for your application. You can set this to any value and retrieve it later.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
For each document, a new text object is allocated by the `TXNNewObject` function and returned in the `oTXNObject` parameter. The object is allocated only if no errors occur, including errors that may occur when reading a file. If there is an error during the allocation process, MLTE frees the text object.

If you are writing a text editing application, you may want to call the `TXNNewObject` function when the application launches (a new document will be displayed) and whenever the user selects New from the File menu.

Many MLTE functions require you to pass a text object; some functions also require the frame ID supplied back to your application in the `oTXNFrameID` parameter of `TXNNewObject`.

Because of how MLTE uses Carbon events internally, the window in which the document is displayed must have the standard event handlers installed. You can do this in one of the following ways:

- When you create the window, add the attribute `kWindowStandardHandlerAttribute` to the window. See *Inside Mac OS X: Window Manager Reference* for more information.
- Call the Carbon Event Manager function `InstallStandardEventHandler` on the window's event target. See *Inside Mac OS X: Handling Carbon Events* for more information.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNOffsetToPoint

Gets the local coordinates of the point that corresponds to a specified offset of a text object. (Deprecated in Mac OS X v10.3. Use `TXNOffsetToHIPoint` (page 65) instead.)

Not recommended.

```
OSStatus TXNOffsetToPoint (
   TXNObject iTXNObject,
   TXNOffset iOffset,
   Point *oPoint
);
```

Parameters

`iTXNObject`
The text object for which you want to obtain the local coordinates of a point.

`iOffset`
An offset value.

`oPoint`
On return, a pointer to the local coordinates of the point that corresponds to the value of the `iOffset` parameter.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
Offsets in MLTE are always character offsets.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNPointToOffset

Gets the offset value that corresponds to a point in local coordinates. (Deprecated in Mac OS X v10.3. Use `TXNHIPointToOffset` (page 61) instead.)

Not recommended.

```
OSStatus TXNPointToOffset (
   TXNObject iTXNObject,
   Point iPoint,
   TXNOffset *oOffset
);
```

Parameters

iTXNObject
The text object for which you want to obtain an offset value.

iPoint
The local coordinates of a point.

oOffset
On return, a pointer to the offset that corresponds to the value of the `iPoint` parameter.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
Offsets in MLTE are always character offsets.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetDataFromFile

Replaces a range of data with the contents of a file. (Deprecated in Mac OS X v10.3. Use `TXNSetDataFromCFURLRef` (page 182) instead.)

Not recommended.

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

Parameters

iTXNObject
The text object that identifies the document in which you want to replace data.

iFileRefNum
The file reference obtained when you opened the file.

iFileType
The file type of the file from which you are getting data. MLTE supports `'RTF '` as well as the file types specified by the constants described in Supported File Types (page 153).

iFileLength
: A value that specifies how much data should be read. This parameter is ignored if the file type is the custom file format (represented by the constant `kTXNTextensionFile`) that MLTE supports. This parameter is useful when your application uses MLTE to read data that is embedded in your application's private file. If you want MLTE to deal with the entire file, set the `iFileLength` parameter to a value of `kTXNEndOffset`.

iStartOffset
: The starting position at which to insert the file into the document. You can use the `TXNGetSelection` function to get the absolute offsets of the current selection. If you want to replace the entire document, then set the `iStartOffset` parameter to 0.

iEndOffset
: The ending position of the range being replaced by the file. You can use the `TXNGetSelection` function to get the absolute offsets of the current selection.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
Your application must open the data fork of the file and set the `iStartOffset` parameter to the appropriate value before you call the `TXNSetDataFromFile` function. If you want to embed MLTE data within private data or other MLTE data, you must set the file position to the appropriate position.

Offsets in MLTE are always character offsets.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetRectBounds

Set the view rectangle and/or the destination rectangle. (Deprecated in Mac OS X v10.3. Use `TXNSetHIRectBounds` (page 78) instead.)

Not recommended.

```
void TXNSetRectBounds (
   TXNObject iTXNObject,
   const Rect *iViewRect,
   const TXNLongRect *iDestinationRect,
   Boolean iUpdate
);
```

Parameters

iTXNObject
: The text object for the current text area.

iViewRect
: A pointer to a `Rect` data structure that contains the new coordinates for the view rectangle. If you do not want to change the view rectangle pass `NULL`.

iDestinationRect
: A pointer to a `TXNLongRect` data structure that contains the new coordinates for the destination rectangle. If you do not want to change the destination rectangle pass `NULL`.

iUpdate
: Pass `true` if you want the text and location of the scroll bars recalculated and redrawn; otherwise pass `false`.

Discussion

You can specify coordinates for one or both rectangles. The view rectangle controls the text you see. The destination rectangle controls how text is laid out. Scroll bars are drawn inside the view rectangle.

If you set the `iViewRect` parameter to a location not currently represented by the scroll bar and you pass `NULL` for the *iDestinationRect* parameter, it becomes impossible to scroll to the left bounds of the destination rectangle. If you want to position the view rectangle inside the destination rectangle, you should supply a custom scrolling callback. See `TXNScrollInfoProcPtr` (page 92).

Availability

Available in Mac OS X v10.1 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNTerminateTextension

Closes the MLTE library. (Deprecated in Mac OS X v10.3. This function is no longer needed.)

Not recommended.

```
void TXNTerminateTextension (
   void
);
```

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

Deprecated in Mac OS X v10.4

DisposeTXNActionKeyMapperUPP

Disposes of the universal procedure pointer (UPP) to your action key mapping callback function. (Deprecated in Mac OS X v10.4. Use `TXNActionNameMapperProcPtr` instead.)

Not recommended.

```
void DisposeTXNActionKeyMapperUPP (
   TXNActionKeyMapperUPP userUPP
);
```

Discussion
See the callback `TXNActionKeyMapperProcPtr` (page 88) for more information.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
`MacTextEditor.h`

InvokeTXNActionKeyMapperUPP

Calls your action key mapping callback function. (Deprecated in Mac OS X v10.4. Use `TXNActionNameMapperProcPtr` instead.)

Not recommended.

```
CFStringRef InvokeTXNActionKeyMapperUPP (
   TXNActionKey actionKey,
   UInt32 commandID,
   TXNActionKeyMapperUPP userUPP
);
```

Return Value
See the Base Services documentation for a description of the `CFStringRef` data type.

Discussion
See the callback `TXNActionKeyMapperProcPtr` (page 88) for more information.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
`MacTextEditor.h`

NewTXNActionKeyMapperUPP

Creates a new universal procedure pointer (UPP) to a callback function that uses your criteria for mapping actions. (Deprecated in Mac OS X v10.4. Use `TXNActionNameMapperProcPtr` instead.)

Not recommended.

```
TXNActionKeyMapperUPP NewTXNActionKeyMapperUPP (
   TXNActionKeyMapperProcPtr userRoutine
);
```

Return Value
A universal procedure pointer.

Discussion

See the callback `TXNActionKeyMapperProcPtr` (page 88) for more information.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`MacTextEditor.h`

TXNCanRedo

Returns whether the most recently undone action is redoable and indicates the type of action that can be redone. (Deprecated in Mac OS X v10.4. Use `TXNCanRedoAction` (page 30) instead.)

Not recommended.

```
Boolean TXNCanRedo (
   TXNObject iTXNObject,
   TXNActionKey *oTXNActionKey
);
```

Parameters

`iTXNObject`

The text object for the document you want to examine.

`oTXNActionKey`

A pointer to a value of type `TXNActionKey`. On return, this value specifies the action that can be redone. See Action Constants (page 106) for a description of possible values. You can use this information to customize the Redo menu item for the specific action to be redone. For example, if the value obtained by `TXNCanRedo` is `kTXNTypingAction`, you can map that value to a string that reads "Redo Typing" on a system localized for U.S. English. MLTE does not perform the mapping your program is responsible for mapping the key to the appropriate localized string you want displayed to the user. Pass `NULL` if you do not want to obtain this information.

Return Value

A `Boolean` value. If `true`, the last command is redoable; otherwise the last command cannot be redone.

Discussion

You can call the `TXNCanRedo` function to determine whether the Redo item in the Edit menu should be enabled.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNCanUndo

Returns whether the most recent action is undoable and provides a value that indicates the type of action than can be undone. (Deprecated in Mac OS X v10.4. Use `TXNCanUndoAction` (page 31) instead.)

Not recommended.

```
Boolean TXNCanUndo (
   TXNObject iTXNObject,
   TXNActionKey *oTXNActionKey
);
```

Parameters

iTXNObject

The text object for the document you want to examine.

oTXNActionKey

A pointer to a value of type `TXNActionKey`. On return, this value identifies the action that can be undone. See Action Constants (page 106) for a description of possible values. You can use this information to customize the Undo menu item for the specific action to be undone. For example, if the value obtained by `TXNCanUndo` is `kTXNTypingAction`, you can map that value to a string that reads "Undo Typing" on a system localized for U.S. English. MLTE does not perform such a mapping your program is responsible for mapping the key to the appropriate localized string you want displayed to the user. Pass `NULL` if you do not wish to obtain this information.

Return Value

A `Boolean` value. If `true`, the last command is undoable; otherwise the last command cannot be undone.

Discussion

You can call `TXNCanUndo` to determine whether the Undo item in the Edit menu should be enabled.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNClearActionChangeCount

Resets the specified action counters to zero. (Deprecated in Mac OS X v10.4. Use `TXNClearCountForActionType` (page 32) instead.)

Not recommended.

```
OSStatus TXNClearActionChangeCount (
   TXNObject iTXNObject,
   TXNCountOptions iOptions
);
```

Parameters

iTXNObject

The text object whose action counter you want to reset.

iOptions

The `TXNCountOptions` to use when resetting the count. See Action Count Masks (page 108) for information on the options you can supply.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion
You can use this function to clear the action counters as needed for your application.

Availability
Available in Mac OS X v10.1 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetActionChangeCount

Retrieves the number of times the specified action or actions have occurred since the count was initialized or cleared. (Deprecated in Mac OS X v10.4. Use `TXNGetCountForActionType` (page 51) instead.)

Not recommended.

```
OSStatus TXNGetActionChangeCount (
   TXNObject iTXNObject,
   TXNCountOptions iOptions,
   ItemCount *oCount
);
```

Parameters

`iTXNObject`
: The text object whose action count you want to retrieve.

`iOptions`
: The `TXNCountOptions` to use when retrieving the count. See Action Count Masks (page 108) for information on the options you can supply.

`oCount`
: On return, a pointer to the action count.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Availability
Available in Mac OS X v10.1 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetFontDefaults

Makes a copy of the font descriptions for a given text object. (Deprecated in Mac OS X v10.4.)

```
OSStatus TXNGetFontDefaults (
   TXNObject iTXNObject,
   ItemCount *ioCount,
   TXNMacOSPreferredFontDescription oFontDefaults[]
);
```

Parameters

`iTXNObject`
: The text object for the document whose default font settings you want to copy.

`ioCount`
: A pointer to a value of type `ItemCount`. You need to call the `TXNGetFontDefaults` function twice (see Discussion). The first time you call the function, pass `NULL`. On return, the `ioCount` parameter specifies the number of font descriptions associated with the text object. The second time you call `TXNGetFontDefaults`, on return the `ioCount` parameter points to the number of font descriptions in the `iFontDefaults` array.

`iFontDefaults`
: An array of `TXNMacOSPreferredFontDescription` structures to be filled. You need to call the `TXNGetFontDefaults` function twice (see Discussion). The first time you call the function pass `NULL`. The second time you call `TXNGetFontDefaults`, you can initialize the `iFontDefaults` array to have the number of elements specified by the `ioCount` parameter. Then on return, the `iFontDefaults` parameter contains the font descriptions for the text object.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
You need to call this function twice: once to get the number of font default descriptions (returned in the `ioCount` parameter), and the second time to get the font default data.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSave

Saves the contents of the document as the file type you specify. (Deprecated in Mac OS X v10.4. Use `TXNWriteRangeToCFURL` (page 86) instead.)

Not recommended.

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

Parameters

iTXNObject

The text object for the active document.

iType

The file type to which the text object should be saved. The type must be `kTXNTextensionFile`, `kTXNTextFile`, or `kTXNUnicodeTextData`. See Supported File Types (page 153) for more information on file type constants.

iResType

The type of resource that should be used to save the style information if the file is being saved as plain text. This parameter is ignored for file types that are not plain text.

iPermanentEncoding

The encoding in which to save the document. If the internal encoding used by MLTE does not match the requested encoding type, the text is translated by the Conversion Manager.

iFileSpecification

A pointer to a variable that specifies the location of the file. This parameter is retained and used in calls to the `TXNRevert` (page 70) function. It is not retained once the text object is deleted or disposed of.

iDataReference

The data fork reference number of the open file.

iResourceReference

The resource fork reference number of the open file. This parameter is ignored if the file type is not `kTXNTextFile`. You can save text without style information by passing `-1` for this parameter.

Return Value

A result code. See "MLTE Result Codes" (page 160).

Discussion

You must first open the file to which you want to save the document. If you are saving the file as plain text and the application has specified a resource type in which to save style attributes, then you must also open the file's resource fork.

MLTE does not move the marker before writing the file. You must make sure the file marker of the opened file is at the position where you want data to be written. Typically, this is position 0, but you can specify any valid file position. This behavior lets you write private data, followed by data that is written by MLTE, which can subsequently be followed by more private data or even another MLTE file.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetDataFromCFURLRef

Replaces a range of data with the contents of a file. (Deprecated in Mac OS X v10.4. Use `TXNReadFromCFURL` (page 67) instead.)

Not recommended.

```
OSStatus TXNSetDataFromCFURLRef (
   TXNObject iTXNObject,
   CFURLRef iURL,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset
);
```

Parameters

iTXNObject
: The text object that identifies the document in which you want to replace data. You can either call the function `TXNCreateObject` to allocate a text object or you can call the function `HITextViewGetTXNObject` (page 23) to obtain the text object associated with an HITextView.

iURL
: A reference to the Core Foundation URL that specifies the file which contains the data you want to add to the object.

iStartOffset
: The starting position at which to insert the file into the document. If you want to replace the current selection, set the `iStartOffset` parameter to `kTXNUseCurrentSelection`. If you want to replace the entire document, set the `iStartOffset` parameter to `0`. Offsets in MLTE are always character offsets.

iEndOffset
: The ending position of the range being replaced by the file. You can use the `TXNGetSelection` function to get the absolute offsets of the current selection. Offsets in MLTE are always character offsets.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
Your application must open the file and set the `iStartOffset` parameter to the appropriate value before you call the `TXNSetDataFromCFURLRef` function. If you want to embed MLTE data within private data or other MLTE data, you must set the file position to the appropriate position.

In the (now deprecated) function `TXNNewObject` your could pass a file reference and MLTE supported functionality to revert back to the original file reference. When you call the function `TXNSetDataFromCFURLRef`, MLTE saves the `CFURLRef`. If you change the contents of the text object and then call the function `TXNRevert`, the document reverts to the contents specified by the saved `CFURLRef`.

Availability
Available in Mac OS X v10.3 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetFontDefaults

Specifies the font descriptions for each script used in a text object. (Deprecated in Mac OS X v10.4.)

```
OSStatus TXNSetFontDefaults (
   TXNObject iTXNObject,
   ItemCount iCount,
   const TXNMacOSPreferredFontDescription iFontDefaults[]
);
```

Parameters

iTXNObject
The text object for the document whose fonts you want to specify.

iCount
The number of font descriptions in the `iFontDefaults` array.

iFontDefaults
An array of `TXNMacOSPreferredFontDescription` structures that contain the font description you want to use for each script.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

Deprecated in Mac OS X v10.5

TXNDisposeFontMenuObject

Disposes of a Font menu object. (Deprecated in Mac OS X v10.5.)

```
OSStatus TXNDisposeFontMenuObject (
   TXNFontMenuObject iTXNFontMenuObject
);
```

Parameters

iTXNFontMenuObject
The Font menu object which you want to dispose of.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
The `TXNDisposeFontMenuObject` function releases the specified Font menu object from memory. Note that `TXNDisposeFontMenuObject` does not dispose of the main Font menu handle that is associated with the Font menu object. You are responsible for disposing of the Font menu handle after calling `TXNDisposeFontMenuObject`. However, `TXNDisposeFontMenuObject` does dispose of any submenus that MLTE creates to support Apple Type Services for Unicode Imaging (ATSUI) fonts.

A good time to call the `TXNDisposeFontMenuObject` function is when your application quits (that is usually when your application no longer needs the Font menu). You can dispose of the Font menu object as part of a "terminate" function you create to do cleanup and termination tasks.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNDoFontMenuSelection

Changes the font of the current selection. (Deprecated in Mac OS X v10.5.)

```
OSStatus TXNDoFontMenuSelection (
   TXNObject iTXNObject,
   TXNFontMenuObject iTXNFontMenuObject,
   SInt16 iMenuID,
   SInt16 iMenuItem
);
```

Parameters

`iTXNObject`
The text object that contains the current selection.

`iTXNFontMenuObject`
The Font menu object that identifies the current Font menu.

`menuID`
The menu ID of the selected menu. You should supply the high 16 bits of the long word obtained from the Menu Manager function `MenuSelect`. You must pass the menu ID because the Font menu may have hierarchical submenus.

`menuItem`
A value that identifies the selected menu item. You should supply the low 16 bits of the long word obtained from the Menu Manager function `MenuSelect`.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
When you receive a mouse-down event in a menu used in your application, you typically call the Menu Manager function `MenuSelect` to determine which menu and menu item the user has chosen. After calling the `MenuSelect` function, you should check whether the mouse-down event occurred in a menu specific to your application, other than the standard menus such as File, Edit, and Font. If the mouse-down event did not occur in a menu specific to your application, you should pass the IDs of the menu and menu item to the

`TXNDoFontMenuSelection` function. If the value you supply in the `iMenuID` parameter identifies the Font menu or one of its submenus, `TXNDoFontMenuSelection` changes the font of the currently selected text to the font that the user selects.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetFontMenuHandle

Gets the Font menu handle that belongs to a Font menu object. (Deprecated in Mac OS X v10.5.)

```
OSStatus TXNGetFontMenuHandle (
   TXNFontMenuObject iTXNFontMenuObject,
   MenuRef *oFontMenuHandle
);
```

Parameters

`iTXNFontMenuObject`

A Font menu object.

`fontMenuHandle`

A pointer to a menu handle. On return, a pointer to the Font menu created when the Font menu object was created. MLTE makes a copy of the menu handle. Your application should not dispose of the menu handle until it disposes of the Font menu object by calling the `TXNDisposeFontMenuObject` (page 183).

Return Value
A result code. See "MLTE Result Codes" (page 160).

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNNewFontMenuObject

Creates a new Font menu object. (Deprecated in Mac OS X v10.5.)

```
OSStatus TXNNewFontMenuObject (
   MenuRef iFontMenuHandle,
   SInt16 iMenuID,
   SInt16 iStartHierMenuID,
   TXNFontMenuObject *oTXNFontMenuObject
);
```

Parameters

iFontMenuHandle

A value of type `MenuRef` obtained by calling the Menu Manager functions `CreateNewMenu`, `NewMenu`, `CreateMenuFromNib`, or `GetMenu`. Before calling `TXNNewFontMenuObject`, initialize the `menuData` field of the handle to specify the menu title, in Pascal string format. You must make sure the string is localized appropriately.

iMenuID

The menu ID of the Font menu.

iStartHierMenuID

The first MenuID to use if any hierarchical menus need to be created. This function calls `SetMenuItemHierarchicalID` to create hierarchical menus, so the value of this parameter must follow the rules for that function. On systems less than Mac OS 8.5, the submenuID must be less than 255. For systems after Mac OS 8.5, the range can be as large as large 32767. However, it is important to remember that `TXNNewFontMenuObject` only uses `iStartHierMenuID` as a starting ID when adding hierarchical menus. Therefore provide plenty of room to increment this value. For example, on a system less than Mac OS 8.5, start at 175. On systems after than Mac OS 8.5, do not use a value more than 32000.

oTXNFontMenuObject

A pointer to a structure of type `TXNFontMenuObject` (page 99). On return, a new Font menu object.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
A good time to call the `TXNNewFontMenuObject` function is when you are preparing to display your menu bar. This function fills the Font menu with font names. Later, you can pass the Font menu object along with a text object to the `TXNDoFontMenuSelection` function which handles all aspects of user interaction with the Font menu.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNPrepareFontMenu

Prepares a Font menu for display. (Deprecated in Mac OS X v10.5.)

```
OSStatus TXNPrepareFontMenu (
   TXNObject iTXNObject,
   TXNFontMenuObject iTXNFontMenuObject
);
```

Parameters

`iTXNObject`
The text object that identifies the document with the Font menu you want to prepare. Pass `NULL` to display an inactive menu (dimmed).

`iTXNFontMenuObject`
A Font menu object.

Return Value
A result code. See "MLTE Result Codes" (page 160).

Discussion
You should call the `TXNPrepareFontMenu` function just before your application opens the Font menu for your user. If the text object's current selection is a single font, MLTE places a check mark next to the menu item for that font.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

Document Revision History

This table describes the changes to *Multilingual Text Engine Reference*.

Date	Notes
2007-02-19	Added documentation for a constant and fixed typographical errors.
2006-07-13	Minor formatting changes.
2006-07-24	Added information on deprecated functions.
2006-01-10	Updated for Mac OS X v10.4.
2004-02-24	Added information to the Discussion for the function `TXNSetRectBounds` (page 174).
2003-07-25	Added information for the data type `TXNLongRect` (page 100).
	Adding documentation for functions available in Mac OS X version 10.3 and later: `TXNCreateObject` (page 35),`TXNAttachObjectToWindowRef` (page 29),`TXNGetWindowRef` (page 60),`TXNDrawObject` (page 42),`TXNSetDataFromCFURLRef` (page 182),`TXNFlattenObjectToCFDataRef` (page 46),`TXNSetHIRectBounds` (page 78),`TXNGetHIRect` (page 54),`TXNSetScrollbarState` (page 79),`TXNHIPointToOffset` (page 61),`TXNOffsetToHIPoint` (page 65),`HITextViewCreate` (page 22),`HITextViewGetTXNObject` (page 23)
	Added documentation for enumerations available in Mac OS X 10.3 and later: `HIObject Class ID` (page 139),`HIObject Control Kind` (page 139),`Draw Items Bits` (page 123),`Draw Items Masks` (page 124),`Rectangle Keys` (page 146)
	Changed documentation for `Supported Frame Types` (page 154).
	Added porting information for functions that are no longer recommended as of Mac OS X version 10.3: `TXNTerminateTextension` (page 175),`TXNNewObject` (page 170), `TXNSetDataFromFile` (page 173),`TXNDraw` (page 167), `TXNIsObjectAttachedToWindow` (page 169), `TXNIsObjectAttachedToSpecificWindow` (page 168),`TXNAttachObjectToWindow` (page 165), `TXNSetRectBounds` (page 174), `TXNGetRectBounds` (page 167),`TXNPointToOffset` (page 172), `TXNOffsetToPoint` (page 172), `TXNSetViewRect` (page 163), `TXNConvertFromPublicScrap` (page 166), `TXNConvertToPublicScrap` (page 166), `TXNActivate` (page 164)
	Moved all functions that are no longer recommended to the section "Not Recommended" (page 18).

Date	Notes
2003-05-01	Fixed typographical errors in the description of the function `TXNGetIndexedRunInfoFromRange`.
2003-04-01	Added additional information on how to set up a read-only text object. See the `TXNNewObject` (page 170) and `Frame Option Bits` (page 133).
2002-12-01	Merged the reference described in the document *Setting Up MLTE to Use Carbon Events*.
	Added the functions `TXNSetRectBounds`, `TXNGetRectBounds`, `TXNRecalcTextLayout`, `TXNScroll`, `TXNRegisterScrollInfoProc`, and the callback `TXNScrollInfoProcPtr`. Added documentation for data types and constants used in these functions.
	Fixed technical and formatting errors.
	Removed the appendix that described the MLTE file format.
2000-05-01	First release of the MLTE Reference.

Index