MacApp 2.0b5 UTEView Release Notes

Russ Wetmore

Overview

This document documents changes and additions to the `UTEView` unit of MacApp. There are several reasons why these changes were desirable:

- To support the new MacApp 2.0 display architecture
- To add support for the new, "styled" TextEdit
- To simplify the unit and add consistency
- To improve its (already substantial) memory management facilities

There are a couple of limitations which prevent `TTEView` from fully conforming to the new MacApp 2.0 display architecture. They are:

- The text in a TextEdit record (`TERec.hText`) is limited to 32K characters
- The height of the entire record must be less than 32K pixels tall.

For these reasons, TTEView objects cannot take advantage of MacApp's 32-bit view coordinates

.

The new TextEdit stresses memory management with more relocatable data structures, and many more temporary objects. TTEView tries its best to assure that any TTECommand is undoable. With version 1.0/1.1, however even with the precautions taken it is still possible to run into out-of-memory conditions that prevent undoing commands. The addition of extra data structures/objects just intensifies the need for new algorithms to assure that any command will be undoable.

Note: In the descriptions that follow, new or changed methods, fields, and parameters are underlined.

Global Constants

`cStyleChange`	`(= 130)`	"Menu" command for style change.
`kUnlimited`	`(= MAXINT)`	The maximum number of characters in the `fText` of a `TTEView` object.
`kWithStyle`	`(= TRUE)`	Parameter to `TTEView.ITEView`.
`kWithoutStyle`	`(= FALSE)`	Parameter to `TTEView.ITEView`.
`kSaveCurrentChars`	`(= TRUE)`	Parameter to `TTECommand.ITECommand`.

The TTEView Class

The following changes have been made for TTEView:

- fFont, fSize and fStyle have been replaced by a single fTextStyle record of type TextStyle. TextStyle is a data structure used by the new TextEdit, and is defined as

```
TextStyle     = RECORD
                   tsFont   : INTEGER;
                   tsFace   : Style;
                   tsSize   : INTEGER;
                   tsColor  : RGBColor;
                END;
```

- TTEView.ITEView has been simplified. Some rarely used parameters have been removed (such as itsKeyCmdNumber and itsMaxChars) and others have been bunched together. A new parameter itsStyleType signifies whether the TERecord is "styled" or not.

- TTEView.Inspect and TTEView.ShowDebugInfo have been replaced by the new Inspector-supported TTEView.Fields.

Fields

fHTE	Handle to the actual TextEdit object.
fText	The text in the TEHandle.
fSavedTEHandle	Saved handle from TENew.
fInset	A rectangle defining the amount of pixels to inset the TERecord's viewRect from view's extent. Thus, fInset defines margins around the edges of the view. Views whose text autowraps should have a bottom margin of zero.
fKeyCmdNumber	Will be used as the string number for "Undo Typing."
fMaxChars	Maximum number of chars to accept into fText. Default is MAXINT; stuff to different value if you want to limit max chars to fewer than MAXINT.
fTypingCommand	The current TTETyping command relating to this object, if any.
fTextStyle	Current style of text.
fJustification	Justfication of text record.
fAcceptsChanges	Set to FALSE to have text which will not accept any change, such as text on the Clipboard, or perhaps received mail.
fStyleType	Set to kWithStyle if record is styled, or kWithOutStyle if the record is not styled.
fAutoWrap	Set to FALSE for line wrapping at carriage returns only.
fFreeText	Determines if fText should be freed on Free.
fSpecsChanged	Something recently happened which could affect font, style, size, and color menu item updating. Should be reset to FALSE when the application has taken appropriate action.
fLastHeight	The last cached value for the height of the record.
fScroller	The scroller in which this view is contained.
fLastPageBreak	The last cached page break. Used when computing page breaks for printing.
fLastLine	The line number of the last page cached page break.

Initializing and Freeing Methods

```
PROCEDURE TTEView.ITEView (itsDocument: TDocument; itsSuperView: TView;
    itsLocation, itsSize: VPoint;
    itsHDeterminer, itsVDeterminer: SizeDeterminer;
    itsInset: Rect; itsTextStyle: TextStyle; itsJustification: INTEGER;
    itsStyleType, itsAutoWrap: BOOLEAN);
```

This method initializes the view; if unsuccessful, exit is via `Failure` mechanism. (Note: The `itsText` parameter, which used to allow you to set your own text at record creation time, has been removed. You should now use the `StuffText` method to set text, and the associated `StuffStyles` method to set saved style information.)

```
PROCEDURE TTEView.IRes (itsDocument: TDocument; itsSuperView: TView;
    VAR itsParams: Ptr); OVERRIDE;
```

This method initializes the view from a `'view'` template.

```
PROCEDURE TTEView.WRes (theResource: ViewRsrcHndl;
    VAR itsParams: Ptr); OVERRIDE;
```

This method writes the object in its present state in `'view'` resource format to the location pointed at by `itsParams`. This is the inverse of the `IRes` method and is included only for completeness. You will rarely need to call it yourself.

```
PROCEDURE TTEView.WriteRes (theResource: ViewRsrcHndl;
    VAR itsParams: Ptr); OVERRIDE;
```

This method is a wrapper method for `WRes`. If you need to write out an object as part of a `'view'` resource, this is the routine you would actually call. It sets up the signature and class name for the template, and then calls `WRes`. You would override this method to provide your own unique class name or signature.

```
PROCEDURE TTEView.Free; OVERRIDE;
```

This method frees the `TERecord` and associated records (and optionally `fText`), and then frees `SELF`.

```
PROCEDURE TTEView.MakeTERecord;
```

This method is called to create the actual `TERecord`. `fHTE` is set to the handle to the `TERecord`. This method is called from `ITEView` and `IRes`.

Command and Menu Manipulation Methods

```
FUNCTION  TTEView.DoKeyCommand (ch: Char; aKeyCode: INTEGER; VAR info:
    EventInfo): TCommand; OVERRIDE;
```

This method handles keystrokes.

```
FUNCTION  TTEView.DoMakeEditCommand (aCmdNumber: CmdNumber): TTECommand;
```

This method makes a command for handling edit menu stuff.

```
FUNCTION  TTEView.DoMakeStyleCommand (aStyle: TextStyle;
    itsCmdNumber: CmdNumber; itsMode: INTEGER): TTEStyleCommand;
```

This method makes a command for changing styles.

```
FUNCTION  TTEView.DoMakeTypingCommand (ch: Char): TTETypingCommand;
```

This method makes a typing command for handling keystrokes.

```
FUNCTION  TTEView.DoMenuCommand (aCmdNumber: CmdNumber): TCommand; OVERRIDE;
```

This method handles menu commands.

```
FUNCTION  TTEView.DoMouseCommand (VAR theMouse: Point; VAR info: EventInfo;
    VAR hysteresis: Point): TCommand; OVERRIDE;
```

This method handles mouse clicks.

```
PROCEDURE TTEView.DoneTyping;
```

This method signifies that no further typing can occur for the current typing command. You need to call this method to resolve the styles arrays being maintained by the `AddCharacter` method.

```
PROCEDURE TTEView.DoSetupMenus; OVERRIDE;
```

This method sets up menus.

```
PROCEDURE TTEView.InstallSelection (wasActive, beActive: BOOLEAN); OVERRIDE;
```

This method is called at activate/deactivate time to get insertion-point blinking right (or to highlight the selection).

Screen Display Methods

```
PROCEDURE TTEView.DoIdle (phase: IdlePhase); OVERRIDE;
```

This method blinks the insertion point.

```
FUNCTION  TTEView.DoSetCursor (localPoint: Point;
    cursorRgn: RgnHandle): BOOLEAN; OVERRIDE;
```

This method sets the cursor to the I-beam (see `TView.DoSetCursor` for `cursorRgn` addition).

```
PROCEDURE TTEView.Draw (area: Rect); OVERRIDE;
```

This method draws the text in a frame or on the printed page.

```
PROCEDURE TTEView.ShowReverted; OVERRIDE;
```

This method makes sure line starts are correct before redisplay.

View Management Methods

```
PROCEDURE TTEView.BeInPort (itsPort: GrafPtr); OVERRIDE;
```

This method tells `TextEdit` about a view once a `grafPort` is established for it.

```
PROCEDURE TTEView.BeInScroller (itsScroller: TScroller); OVERRIDE;
```

This method gives us a chance to set the scroll parameters.

```
PROCEDURE TTEView.CalcMinSize (VAR minSize: VPoint); OVERRIDE;
```

This method computes the *minimum* size of the view. (This method was originally called `CalcMinExtent`.)

```
PROCEDURE TTEView.CalcRealHeight: INTEGER;
```

This method computes the *logical* height of the view (taking into account the last line if the last character is a carriage return).

```
PROCEDURE TTEView.ComputeSize (VAR newSize: VPoint); OVERRIDE;
```

This method computes the *actual* size of the view. (This method was originally called `ComputeExtent`.)

```
PROCEDURE TTEView.Resize (width, height: VCoordinate; invalidate: BOOLEAN);
    OVERRIDE;
```

This method installs the new size for both MacApp and TextEdit. (This method was originally called `SetExtent`.)

```
PROCEDURE TTEView.SynchView (redraw: BOOLEAN);
```

This method keeps view metrics in sync after a TextEdit operation.

Clipboard-Related Methods

NOTE: These methods apply only to a `TEView` installed as the view in the Clipboard.

```
FUNCTION  TTEView.ContainsClipType (aType: ResType): BOOLEAN; OVERRIDE;
```

This method determines whether the indicated Clipboard type can be produced by the view.

```
FUNCTION  TTEView.GivePasteData (aDataHandle: Handle;
    dataType: ResType): LONGINT; OVERRIDE;
```

This method supplies data to be pasted by some requesting Paste command.

```
PROCEDURE TTEView.WriteToDeskScrap;  OVERRIDE;
```

This method produces `TEXT` data for the Desk Scrap when called upon to do so.

Miscellaneous Methods

```
PROCEDURE TTEView.AutoScrolling (doScrolling: BOOLEAN);
```

This method sets the `TERecord` auto-scrolling to the given value via `TEAutoScroll`.

```
PROCEDURE TTEView.CalcSelLoc (VAR selLocation: Point);
```

This method calculates the starting location of the current selection.

```
PROCEDURE TTEView.ChangeWrap (newAutoWrap, redraw: BOOLEAN);
```

This method changes auto-wrapping behavior to given, redrawing if requested.

```
FUNCTION  TTEView.ContinuousStyle (firstChar, lastChar: INTEGER;
    VAR mode: INTEGER; VAR aStyle: TextStyle): BOOLEAN;
```

This method returns `TRUE` if the style for the characters from `firstChar` to `lastChar` is wholly continuous. The style attributes to check are specified by `mode`, which has the same meaning as in the TextEdit call `TESetStyle`. Attributes that are continuous are returned in `aStyle`, with the appropriate bits set in `mode`. Font style attribute bits are individually checked across the selection range, and they are returned set to 1 if that attribute is continuous.

```
PROCEDURE TTEView.ExtractStyles (VAR theStyles: TEStyleHandle;
    VAR theElements: STHandle);
```

This method extracts all style information from the TextEdit record. Note that line heights can be recalculated, and as such don't need to be stored.

```
FUNCTION  TTEView.ExtractText: Handle;
```

This method extracts all text from TextEdit record.

```
PROCEDURE TTEView.RecalcText;
```

This method tells TextEdit to recompute linestarts.

```
PROCEDURE TTEView.ScrollSelectionIntoView;
```

This method scrolls the selection into view. It is called after commands are done or undone.

```
PROCEDURE TTEView.SetJustification (newJust: INTEGER; redraw: BOOLEAN);
```

This method sets `TERecord` justification to the given value, and redraws if requested.

```
PROCEDURE TTEView.SetOneStyle (theStart, theEnd, theMode: INTEGER;
    theStyle: TextStyle);
```

This method sets the given range in the record to the given `TextStyle`. If the record is unstyled, the change will affect the entire record. `mode` has the same meaning as used in `ContinuousStyle`.

```
PROCEDURE TTEView.SetText (theText: Str255);
```

This method is a convenience routine that saves you the effort of creating a handle just to pass to `StuffText`. Note that if `fSavedTEHandle` already holds a saved handle, it will be disposed before `StuffText` is called.

```
FUNCTION  TTEView.SpaceForStyles (rangeStart, rangeEnd: LONGINT): BOOLEAN;
```

This method returns `TRUE` if there is enough memory to hold the styles over the given range in the `TERecord` (in scrap format).

```
PROCEDURE TTEView.StuffStyles (theStyles: TEStyleHandle; theElements:
    STHandle);
```

This method installs style information, presumably fetched from disk, into the TextEdit record.

```
PROCEDURE TTEView.StuffText (theText: Handle);
```

This method installs text into the `TERecord`. Note that this destroys any existing style information. `StuffStyles` should be called if styles are to be attached to this text information.

```
PROCEDURE TTEView.StuffTERects (newTERect: Rect);
```

This method installs `newTERect` as the destination and view rectangles of the TextEdit record.

Printing-Related Methods

```
FUNCTION  TTEView.DoBreakFollowing (vhs: VHSelect; prevBreak: VCoordinate;
    VAR automatic: BOOLEAN): VCoordinate; OVERRIDE;
```

This method computes the next page break following `prevBreak` in direction given by `vhs`.

```
PROCEDURE TTEView.DoCalcViewPerPage (VAR viewPerPage: VPoint); OVERRIDE;
```

This method computes how much of the view is to be allocated to each printed page

```
PROCEDURE TTEView.DoSetPageOffset (coord: VPoint); OVERRIDE;
```

At printing time, this method determines the location of the *interior* (that is, the body) of the page.

```
PROCEDURE TTEView.GetPrintExtent (VAR printExtent: VRect); OVERRIDE;
```

Returns the view's extent inset by `fInset` so that the margins provided by `fInset` are not included when printing.

Debugging Methods

```
PROCEDURE TTEView.IdentifySoftware; OVERRIDE;
```

This method tells the compile date of this unit.

```
PROCEDURE TTEView.Fields (PROCEDURE DoToField (fieldName: Str255;
    fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
```

This method displays fields of the object for debugging purposes.

The TTECommand Class

`TTECommand` used to consist of methods designed to handle single runs of plain text, the assumption being that a single style affected an entire text record. The introduction of styles, and their particular implementation in TextEdit presented problems not only in capsulating changes, but also in quantitatively measuring them.

Most of its present methods have been rewritten to incorporate an understanding of styles in general, and of the unique format of those styles on the desk scrap.

The `saveCurrentChars` parameter of `TTECommand.ITECommand` has been removed, as it is almost always passed as `TRUE`.

Fields

`fTEView`	The `TEView` operated on.
`fHTE`	Same as `fTEView`'s `fHTE`; duplicated for code efficiency.
`fOldStart` `fOldEnd`	The beginning and ending positions of the selection at the moment just before the command was done.
`fOldText`	If `fOldStart = fOldEnd`, i.e., if old selection had been an insertion point, this will be `NIL`. Otherwise, provides a temporary home for the characters comprising the old selection.
`fOldStyles`	Same format as scrap `'styl'` record, describing format of text in `fOldText`. Will never be `NIL`. Always contains at least one style.
`fNewStart` `fNewEnd`	The beginning and ending locations in the text of the new text that is added by the command, if any.
`fNewText`	A Handle to the characters added by the command.
`fNewStyles`	Describes format of text in `fNewText`.
`fPadding`	Handle to fill size between new and old. This (almost) insures that we can undo and redo.
`fTextPad`	Size difference between New and Old text (originally called fDNewOld).
`fStylePad`	Size difference between New and Old styles.

Initializing and Freeing Methods

```
PROCEDURE TTECommand.ITECommand (itsTEView: TTEView; itsCmdNumber: CmdNumber;
    itsSaveText: BOOLEAN);
```

This method initializes the command; if unsuccessful, exit is via `Failure` mechanism.

```
PROCEDURE TTECommand.Free; OVERRIDE;
```

This method frees the text handles holding information needed for Undo/Redo, then frees `SELF`.

Command Execution Methods

```
PROCEDURE TTECommand.DoIt; OVERRIDE;
```

This method focuses, then calls `DoMainFunction`.

```
PROCEDURE TTECommand.RedoIt; OVERRIDE;
```

This method focuses, then calls `RestoreSelection` to get selection right, then calls `DoMainFunction` to reinstate the changes done in `DoIt` which were undone by a preceding `UndoIt`.

```
PROCEDURE TTECommand.UndoIt; OVERRIDE;
```

This method focuses, then dispatches to `TTECommand.RemoveAdditions`, `TTECommand.ReviveDeletions`, and `TTECommand.RestoreSelection` to undo the command.

Command Execution Utilities

```
PROCEDURE TTECommand.BanishOldText;
```

This method removes text that was selected at the outset of the Do phase.

```
PROCEDURE TTECommand.DoMainFunction;
```

This method dispatches to the relevant methods for Do and Redo phases.

```
PROCEDURE TTECommand.InstallNewText;
```

This method installs the new text.

```
PROCEDURE TTECommand.RemoveAdditions;
```

This method removes any characters which were added by the Do phase of the command.

```
PROCEDURE TTECommand.RestoreSelection;
```

This method sets the selection to be what it was just before the Do phase of the command was performed.

```
PROCEDURE TTECommand.ReviveDeletions;
```

This method brings back the characters which were removed during the Do phase.

Debugging Methods

```
PROCEDURE TTECommand.Fields (PROCEDURE DoToField (fieldName: Str255;
    fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
```

This method displays the fields of the object for debugging purposes.

The TTECutCopyCommand Class

Externally, these routines remain the same. Internally, they have been vastly rewritten to account for styles.

Fields

`fClipCreated`	Clipboard view created OK.

Initializing and Freeing Methods

```
PROCEDURE TTECutCopyCommand.ITECutCopyCommand (itsTEView: TTEView;
    itsCmdNumber: CmdNumber);
```

This method initializes the command.

```
PROCEDURE TTECutCopyCommand.Free; OVERRIDE;
```

This method frees the command.

Command Execution Methods

```
PROCEDURE TTECutCopyCommand.DoIt; OVERRIDE;
```

This method launches a `TTEView` for installation in the Clipboard, copies the current text and styles to it, and then calls `DoMainFunction`.

Command Execution Utilities

```
PROCEDURE TTECutCopyCommand.ReviveDeletions; OVERRIDE;
```

This method brings back the characters which were removed during the Do phase; called during Undo phase.

The TTEPasteCommand Class

There is only one method in this subclass, which remains the same as before externally.

Initializing and Freeing Methods

```
PROCEDURE TTEPasteCommand.ITEPasteCommand (itsTEView: TTEView);
```

This method initializes the command; if unsuccessful, exit is via the `Failure` mechanism.

The TTEStyleCommand Class

One new, additional descendent of TTECommand, named TTEStyleCommand, has been added to handle changes of style in a record.

Fields

`fMode`	Mode for style change.
`fOldTextStyle`	The original text style.
`fNewTextStyle`	What we're replacing it with.

Initializing and Freeing Methods

```
PROCEDURE TTEStyleCommand.ITEStyleCommand (itsTEView: TTEView;
    itsNewStyle: TextStyle);
```

This method initializes the command; if unsuccessful, signalled by `Failure` mechanism.

Command Execution Methods

```
PROCEDURE TTEStyleCommand.DoIt; OVERRIDE;
```

This method applies style over current selection range.

```
PROCEDURE TTEStyleCommand.RedoIt; OVERRIDE;
```

This method redoes selected style change.

```
PROCEDURE TTEStyleCommand.UndoIt; OVERRIDE;
```

This method undoes previously selected style change.

Command Execution Utilities

```
PROCEDURE TTEStyleCommand.InstallOneStyle (newStyl: TextStyle);
```

This method installs one style over entire record. Called for old TextEdit, and during the Do and Redo phases using styled TextEdit.

```
PROCEDURE TTEStyleCommand.InstallManyStyles (newStyls: StScrpHandle);
```

This method install sstyles over current selection range. Called during the Undo phase using styled TextEdit.

Debugging Methods

```
PROCEDURE TTEStyleCommand.Fields (PROCEDURE DoToField (fieldName: Str255;
    fieldAddr: Ptr;  fieldType: INTEGER)); OVERRIDE;
```

This method displays fields of the object for debugging purposes.

The TTETypingCommand Class

The `TTETypingCommand` class has been rewritten to account for styles. TextEdit has no built-in provision for extending nonvisible selections, so maintaining an undoable record of overtyped or stricken characters has been added to the `AddCharacter` method.

`AddCharacter` has been broken out into its three actions: adding a character (`DoNormalChar`), backspacing to the right of the original selection (`BkSpcRight`), and backspacing to the left of the original selection (`BkSpcLeft`).

Fields

`fCompleted`	Indicates whether further keystrokes will be extensions to this command (`FALSE`) or whether the command has already been completed (`TRUE`).
`fFirstChar`	First character typed.

Initializing and Freeing Methods

```
PROCEDURE TTETypingCommand.ITETypingCommand (itsTEView: TTEView;
    itsFirstChar: Char);
```

This method initializse the command; if not successful, exit is via `Failure` mechanism.

```
PROCEDURE TTETypingCommand.Free; OVERRIDE;
```

This method deallocates the command and dependent structures.

Command Execution Methods

```
PROCEDURE TTETypingCommand.DoIt; OVERRIDE;
```

Calls AddCharacter to add the first character for this command.

```
PROCEDURE TTETypingCommand.UndoIt; OVERRIDE;
```

This method processes an Undo Typing command.

Command Execution Utilities

```
PROCEDURE TTETypingCommand.DoNormalChar (aChar: Char);
```

This method handles any typed character *except* a backspace.

```
PROCEDURE TTETypingCommand.BkSpcLeft (theText: Handle; curStart: INTEGER);
```

This method handles backspaces to the *left* of the original selection.

```
PROCEDURE TTETypingCommand.BkSpcRight (theText: Handle; curStart: INTEGER
```

This method handles backspaces to the *right* of the original selection.

```
PROCEDURE TTETypingCommand.AddCharacter (aChar: Char);
```

This method adds another character to an already-launched `TTETypingCommand`.

```
PROCEDURE TTETypingCommand.CompleteTyping;
```

Called when no more typing is allowed for this command and fixes up the stype scrap, if any.

Debugging Methods

```
PROCEDURE TTETypingCommand.Fields (PROCEDURE DoToField(fieldName: Str255;
    fieldAddr: Ptr; fieldType: INTEGER)); OVERRIDE;
```

This method displays fields of the object for debugging purposes.

Global references

```
PROCEDURE InitUTEView;
```

This procedure initializes the `TTEView` unit. This procedure must be called before a `TTEView` can be created from a `'view'` template.

```
FUNCTION  AutoscrollTEView: BOOLEAN;
```

This routine is called from TextEdit when autoscrolling is needed.

```
PROCEDURE SetSelect (theStart, theEnd: INTEGER; hTE: TEHandle);
```

For those times when we don't want the selection range hilited when changed.

```
PROCEDURE DumpTERecord (aTEH: TEHandle);
```

This method writes salient information about the `TERecord` out to the Debug window.

```
PROCEDURE DumpTTECommand (theTTECommand: TTECommand);
```

This method writes extensive information about the given `TTECommand` to the Debug window.