New Technical Notes
Macintosh
®
Developer Support

TextEdit Technicalities

Text **M.TE.TextEditTech**

Revised by: Mary Burke April 1990
Written by: Mary Burke February 1990

This Technical Note discusses some areas in TextEdit that have not previously been clearly documented.
Changes since February 1990: Added a note about the changes in TextEdit for System Software 6.0.5, documented the low-memory global `TESysJust`, clarified information about text direction and `_TESetJust`, discussed problems with the `SetWordBreak` routine along with a solution to work around it, and described the differences in dialog text item behavior.

TextEdit in 6.0.5

In addition to all the features of earlier versions, TextEdit 3.0 now allows you to take advantage of the Script Manager's handling of systems with more than one script system installed. TextEdit uses the Script Manager to support such systems and now exhibits the correct behavior for editing and displaying text in multiple styles and different scripts. Multiple scripts can even exist on a single line due to TextEdit's use of the Script Manager. The new version of TextEdit in 6.0.5:

- handles mixed-directional text
- synchronizes keyboards and fonts
- handles double-byte characters
- determines word boundaries and line breaks
- provides outline highlighting in the background
- buffers text for performance improvements
- permits left justification in right-to-left directional scripts
- customizes word breaking
- customizes measuring

Refer to the TextEdit chapter in *Inside Macintosh*, Volume VI, for detailed documentation on TextEdit 3.0.

The LineStarts Array and nLines

The `LineStarts` array is a field in a TextEdit record that contains the offset position of the first character of **each** line. This array has the following boundary conditions:

- It is a zero-based array.
- The last entry in the array must have the same value as `teLength`.
- The maximum number of entries is 16,000.

To determine the length of a line you can use the information contained in the `lineStarts` array and `nLines`. For example, if you want to determine the length of line n, subtract the value contained in entry n of the array from the value in the entry (n+1):

```
   lengthOfLineN := myTE^^.lineStarts[n+1] - myTE^^.lineStarts[n];
```

The terminating condition for this measurement is when `n = nLines + 1`. It is important not to change the information contained in the array.

TESysJust

`TESysJust` is a low-memory global that specifies the system justification. The default value of this global is normally based on the system script. It is -1 when a system's default line direction is right to left, and 0 for a default left-to-right line direction. Applications may change the value using the Script Manager routine `SetSysJust`; however, these applications should save the current value before using it and restore it before exiting the application or processing a MultiFinder suspend event. The current value may be obtained using the Script Manager routine `GetSysJust`.

Forcing Text Direction

The original TextEdit documentation introduced `_TESetJust` with three possible choices for justification: `teJustLeft` (0), `teJustCenter` (1), and `teJustRight` (-1). These choices are appropriate for script systems that are read from left to right. However, in script systems that are read from right to left, text is incorrectly displayed as left justified in dialog boxes and in other areas of applications where users cannot explicitly set the justification. To fix this problem, the behavior of `teJustLeft` has changed to match the line direction of the system in use, which is the value stored in `TESysJust`. Another constant has been added to allow an application to force left justification: `teForceLeft` (-2). This constant has been available for some time, but it has not been documented until now. If your application does not allow the user to change the justification, then it should use `teJustLeft`; if it does, then it should use `teForceLeft` for left justification.

A Little More on Redraw in _TESetStyle

If the `redraw` parameter used in `_TESetStyle` is `FALSE`, line breaks, line heights, and line ascents are not recalculated. Therefore a succeeding call to a routine using any of this information does not reflect the new style information. For example, a call to `_TEGetHeight` (which returns a total height between two specified lines) uses the line height set previous to the `_TESetStyle` call. A call to `_TECalText` is necessary to update this information. If `redraw` is `TRUE`, the current style information is reflected. This behavior also holds for the `redraw` parameter in `_TEReplaceStyle`.

TEDispatchRec

There is currently space reserved for four documented hooks in the `TEDispatchRec`: `TEEolHook`, `TEWidthHook`, `TEDrawHook` and `TEHitTestHook`. The space beyond these hooks is reserved, and any attempt to use this private area results in corrupted TextEdit data.

Custom Word Breaks

A problem exists in one of TextEdit's advanced procedures, `SetWordBreak`. The current glue code does not preserve the state of the registers correctly; however, the solution is fairly simple. Instead of calling `SetWordBreak` and passing a pointer to your custom word break routine, pass the pointer to your external glue which should call your custom word break routine. Following is the glue code that correctly handles the registers:

```
WordBreakProc    PROC        EXPORT

                IMPORT      MYWORDBREAK         ;Must be uppercase here
                MOVEM.L     D1-D2/A1,-(SP)
                CLR.W       -(SP)               ;Space for result
                MOVE.L      A0,-(SP)            ;Move the ptr to stack
                MOVE.W      D0,-(SP)            ;Move the charpos to Stack
                JSR         MYWORDBREAK
                MOVE.W      (SP)+,D0            ;Set Z bit
                MOVEM.L     (SP)+,D1-D2/A1
                RTS

                ENDP
```

An external declaration is also necessary:

```
   FUNCTION WordBreakProc( text: Ptr; charPos: INTEGER ) : BOOLEAN; EXTERNAL;
```

as is the function itself. One thing that should be noted is that it is not really necessary to have `MyWordBreak` boolean, but rather to have the `Z` bit set properly. The result of the function should be zero when you do not want a break; otherwise, a non-zero value indicates a break is desired.

```
   FUNCTION MyWordBreak( text : Ptr; charPos : INTEGER ) : INTEGER;
   { Your word break code here. }
```

For more information, refer to the TextEdit chapter of *Inside Macintosh*, Volume I-380.

Static and Editable Text

The Dialog Manager depends on TextEdit to display text in dialog boxes. For an editable text field, the Dialog Manager simply calls `_TEUpdate`. Before making this call, it may double the width of the rectangle to contain the text if the height of the rectangle is sufficient for only one line and the line direction specified by `TESysJust` is left to right. In this case, the Dialog Manager extends the rectangle on the right. Note, however, this does not occur when your line direction is right to left.

For static text items, `_TextBox` is used instead. When the display rectangle is not large enough. `_TextBox` clips the text to the size of the specified rectangle. To avoid the clipping problem, simply make the display rectangle larger. If your dialog box contains both static and editable text items, the difference in the text handling may appear inconsistent.

Further Reference:

- *Inside Macintosh*, Volumes I,V & VI, TextEdit
- *Inside Macintosh*, Volume V, The Script Manager
- *Inside Macintosh,* Volume I, The Dialog Manager
- M.TE.TestEditChanges