Customizing TextEdit's Features

To customize **TextEdit**'s capabilities to the specifications of your own application, you can replace some of **TextEdit**'s routines with routines of your own that accomplish the same function. To allow you to do this, **TextEdit** supplies hooks, fields in which you can store the address to a routine if you require different behavior from that provided by **TextEdit**. Normally, you use **TextEdit**'s standard or default routines whose addresses are contained in these fields in the TextEdit Record. To override these routines, you can place the address of your hook routine in the appropriate field by using the **TECustomHook** procedure. See **TECustomHook** for details.

Warning: If you use any of the TextEdit hooks to override default **TextEdit** behavior, the results may no longer be **Script Manager**-compatible. You must determine whether more than one script system is installed before replacing a **TextEdit** routine with an alternate routine.

Also, before placing the address of your routine in the TextEdit dispatch record (defined by the TEDispatchRec data type), you should strip the addresses using the **StripAddress** function of the **Operating System Utilities** to guarantee that your application is 32-bit clean.

Measuring the Width of Components of a Line

Earlier versions of **TextEdit** used the hook WIDTHHook any time the width of various components of a line was measured. **TextEdit** now uses three hooks-nWIDTHHook, TextWidthHook, and WIDTHHook-to measure the width of various components of a line. The hook nWIDTHHook lets you replace **TextEdit**'s new measuring routine for non-Roman script systems. **TextWidthHook** allows you to replace all the new calls to the QuickDraw **TextWidth** function in **TextEdit** with your own measuring routine. WIDTHHook retains its original measuring function to provide backward compatibility for your applications. See **TextEdit Width Hooks** for details.

Defining Word Boundaries

TextEdit provides a higher-level hook, **TEFindWord**, that allows you to customize word breaking. **TextEdit** now disregards the wordBreak hook on non-Roman script systems and only uses it on system software with only the Roman Script System installed if an application has supplied an alternate routine in the hook. See **wordBreak Hook** for details.

Controlling Outline Highlighting, Text Buffering, and Inline Input

TextEdit provides outline highlighting for inactive text. This highlighting is similar to the behavior of MPW selections. **TextEdit** also supplies text buffering for performance improvements. Finally, support for inline input for double-byte script systems is provided with **TextEdit**. This support includes several new features for inline input and the capability to disable inline input. All these features are controlled with the **TEFeatureFlag** function. See **TEFeatureFlag** for details.

Setting Left Alignment for Right-to-Left Directional Scripts

Prior to this version of **TextEdit**, the **TESetJust** procedure provided three possible choices for alignment in its just parameter: the Justification Modes are 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 aligned in dialog boxes and in other areas of applications where users cannot explicitly set the alignment.

An additional constant for the just parameter to the **TESetJust** procedure allows you to specify left alignment if the primary line direction is right to left (that is, TESysJust = -1). This value for the just parameter is teForceLeft (-2).

The behavior of the constant teJustLeft makes alignment occur in the primary line direction specified by TESysJust. Therefore, if you use teJustLeft when the line direction is right to left, right alignment takes place as it does when you use the value teJustRight.

If your application does not allow the user to change the alignment, then it should use teJustLeft; if it does allow the user to change the alignment, then it should use teForceLeft for left alignment.

Note: **TextEdit** does not support fully justified alignment. The **Script Manager** supplies routines you can use to provide support for justified text in your applications. See **Worldwide Software Overview** for details.

Because of the conflict between the names of the just parameter's constants and their effects within **TextEdit**, new names have been provided as shown in **TESetJust**.