TECustomHook Install custom handlers for TextEdit bottleneck routines

#include <TextEdit.h> **TextEdit**

TECustomHook(*which, addr, hTE* **);**

TEIntHook	*which*	0=intEOLHook, 1=intDrawHook, et al.
ProcPtr	***addr*	address of function pointer to install; receives old hook
TEHandle	*hTE* ;	edit record to modify

The **TECustomHook** procedure lets your application customize the features of **TextEdit** by setting the TextEdit hooks.

TECustomHook installs a custom routine for **TextEdit** EOL handling, text drawing, text measuring, and hit-testing. It returns the address of the original handler to provide a way to chain functions.

which specifies which bottleneck routine to install. It is one of:

intEOLHook	0	Calculates text width
intDrawHook	1	Draws text
intWidthHook	2	Measures text
intHitTestHook	3	See which character corresponds to a screen pixel
intNWidthHook	6	Width measurement hook **nWIDTHHook**
intTextWidthHook	7	Width measurement hook **TextWidthHook**

addr is the address of a ProcPtr. On entry, it is the address of your custom routine to take over the function identified by *whichHook*. Upon return, it will contain the address of the routine that normally handles the function.

hTE is a handle leading to an edit record created via **TEStylNew**.

You specify your customized hook in the *addr* parameter. When **TECustomHook** returns, the *addr* parameter contains the address of the previous hook specified by the which parameter. This address is returned so that hooks can be daisy-chained. The two new hooks, nWIDTHHook and TextWidthHook, specified by the intNWidthHook and intTextWidthHook constants, are described in **“**TextEdit Width Hooks**”**.

Two integer fields of the edit record, not used for their original purposes but still named recalBack and recalLines, combine to hold a handle to the TextEdit dispatch record, which contains a list of TextEdit hooks. (See the figure in the TextEdit Data Structures description for an illustration of the edit record, the dispatch record, and all the **TextEdit** data structures.) Each edit record has its own set of such routines to provide for maximum flexibility. You should always use the **TECustomHook** procedure to change these hooks instead of modifying the edit record directly.

Warning: Do not simply copy the recalBack and recalLines fields to another edit record. If you do, a duplicate handle to the initial TextEdit dispatch record is stored in recalBack and recalLines in your copy of the record. When one of the edit records is disposed, the handle stored in the copy becomes invalid, and **TextEdit** can crash if the copy is used.

EOLHook, WIDTHHook, nWIDTHHook, TextWidthHook, DRAWHook, and

HITTESTHook are fields into the TextEdit dispatch record and are described in the next sections.

Note: When you replace these hooks, note that all registers except those specified as containing return values must be preserved. Registers A3 and A4 contain a pointer and a handle, respectively, to the edit record. You can obtain line start positions from the lineStarts array in the edit record. A5 is always valid.

Note: The **TextBox** procedure only uses these hooks when it needs to allocate an edit record.

Returns: none

Notes: In the new-format edit record, the fields TERec.recalBack and recalLines contain a handle to a list of **TextEdit** bottleneck routines. **TECustomHook** provides a means to install your own routines to add functionality (such as TAB-processing or sub/superscripted characters) to a particular edit record.

The intercepted hooks apply only to hTE, and not to other edit records. When you dispose the edit record (via **TEDispose**), the Handle stored in recalBack and recalLines gets deallocated; thus be wary of duplicating an edit record (instead, use **TEStylNew** and **TECustomHook** again).

Remember that *procAddr* is a pointer to a ProcPtr, and receives return information. For instance:

```
extern pascal MyDrawHook();          /* asm routine */
ProcPtr    addr;

hookAddr = MyDrawHook;
TECustomHook( intDrawHook, &addr, hTE );
/* now 'addr' contains the address of the previous handler */
```