

Technical Note TE525

TextEdit Q&As

CONTENTS

[Downloadables](#)

This Technical Note contains a collection of archived Q&As relating to a specific topic - questions sent the Developer Support Center (DSC) along with answers from the DSC engineers. Current Q&A's can be found on the [Macintosh Technical Q&A's web site](#).

[Oct 01 1990]

Introduction

SetStylHandle bug and workaround

`SetStylHandle` appears to dispose of only the `TEStyleHandle` in the TextEdit record and not the handles in the `TEStyleRec` in system software version 7.0.1. Since my application calls `SetStylHandle` often, I'm leaving lots of little handles around in my heap. Can I safely dispose of the handles in the current `TEStyleRec` before calling `SetStylHandle`?

—

`SetStylHandle` does indeed orphan the related handles in your heap, a potential problem for applications using `SetStylHandle`. This won't be fixed in the foreseeable future. Your application will be fine disposing of the handles itself. Here's a routine you can use instead of `SetStylHandle`; it correctly disposes of the substructures in the `TEStyleRec` before calling `SetStylHandle`:

```
void FixedSetStylHandle(TEStyleHandle newstyle, TEHandle hte)
/*
 * This function avoids orphaning the substructures of a
 * TEStyleRec by disposing of them before calling SetStylHandle.
 */
{
    register TEStyleHandle oldstyle;
    /* Get the old-style handle so that we can clean it up. */
    oldstyle = GetStylHandle(hte);
    /* Dispose of the substructures first. */
    DisposeHandle((**oldstyle).styleTab);
    DisposeHandle((**oldstyle).lhTab);
    DisposeHandle((**(**oldstyle).nullStyle).nullScrap);
    DisposeHandle((**oldstyle).nullStyle);
    /* Now we can install the new style. */
    SetStylHandle(newstyle, hte);
}
```

[Back to top](#)

Extending TextEdit beyond 32K limit

Date Written: 9/16/92

Last reviewed: 11/1/92

How do I get around the 32K limit in TextEdit? For example, if I have a 64K text file and I want to read it in and display it in a window, I can only read the first 32K. How do I make the text display smoothly when I need to read the next block of text in?

Unfortunately, there's no supported way of doing this. In the early days, TextEdit was designed for use primarily with the Dialog Manager, for displaying and editing small amounts of text at a time. Since then, many developers have used it, or wanted to use it for much more than it was intended. See the Technical Note "[Don't Abuse the Managers](#)," available on the Developer's CD (path: Technical Documentation: Macintosh Technical Notes: Overview: Managerial Abuse).

The Word Solution Engine by DataPak is a replacement for TextEdit that allows you to go beyond the 32K limit, as well as including many other features. Other packages are available as well. Check the Redgate Macintosh Directory on AppleLink or publications such as *MacTutor* magazine to find them.

[Back to top](#)

Controlling TEIdle's insertion point color

Date Written: 7/6/92

Last reviewed: 6/14/93

Is it possible to control the color of the insertion point when I use `TEIdle`? If I have a light blue background color, the caret is always the invert of that background color and doesn't have enough contrast to see it. I want to make the color of the caret black if possible.

As is often the case, some of the seemingly simplest questions tend to yield the most interesting answers. Succinctly, your question stabs at the wobbly knees of QuickDraw's color model. Sample code to change the color of the caret in a TextEdit record is in the Snippets folder. Here's how to implement the color caret.

You can use the `caretHook` field of the TextEdit record (IM I-379). Basically, this procedure is called repeatedly by `TEIdle` to draw the caret. The function calls `InvertRect` each time. It draws the caret, and then it erases the caret. Unfortunately, this function needs to be implemented in assembly. So there are a few gotchas:

- The pointer to the rectangle for the caret is passed on the stack (not the whole rectangle).
- The function `InvertRect` becomes `_InverRect` in assembly.
- You can only use A4 for scratch (A0 and A1 are blown away by `PaintRect`), and then only if you saved and restore it.
- Make sure you pop off the rectangle pointer if you do not use it.

The default system function is simply `_InverRect` because the rectangle pointer is on the stack and `_InverRect` pops it off. For color, inverting is not the best idea, as you have noted. Inversion, if it isn't black and white, does not give you the most contrast. Thus, we need another idea.

But, remember, we have a requirement to paint once, and then erase on the next pass. Inversion works nicely this way. To meet this requirement, use Xor mode with `PaintRect`. As you know, Xor mode is "or" mode with the double 1s being set to 0 instead of 1. This trait is ideal in your situation because, always having the same caret pattern, drawing a second time obliterates the previous drawing of the caret. Thus, we get the blinking behavior. And, with the Macintosh II and Color QuickDraw, we get these cool things called pixel patterns (IM V-55). You can use your own pixel pattern as the pen pattern, bring it in from the resource, and then use `PaintRect` in the `patXor` mode to achieve the desired effect. Done, right?

Unfortunately, here we run into a major flaw of QuickDraw: It is based on two color models, indexed and direct. And, arithmetic operations are performed on the pixel values, and not the colors. In the indexed model, a pixel value of 0x00 is white. (The pixel value in the indexed color model is really an index into a color table.) If you Xor it with 0xFF, you'll get 0xFF, or black. If you Xor 0xFF with 0xFF, you'll get 0x00 or white again. On the other hand (in the direct color model), a pixel value of 0x00 is black. If you Xor it with 0xFF, you'll get 0xFF, or white. This is a problem.

So, you'll need to perform conditional caret coloring (sounds like a disease). Depending on the depth, you'll need to use a different pixel pattern. One good way is to use `DeviceLoop` (*Inside Macintosh* Volume VI, pages 21-23). Though it's included with System 7.0, there's a simulated one as described in the article "Multiple Screens Revealed" on page 57 of *develop* #10. `DeviceLoop` lets you provide a routine that is called for each screen that the drawing area touches. In this case, the drawing routine is your `PaintRect` routine for the caret. This drawing routine gets passed the depth of the screen. So, based on the depth of the screen, you can use a different pixel pattern to draw with. For pixel depth of 32 or direct color mode, you can use one pattern, and for the rest you can use a different pattern.

Note:

The drawing routine used by `DeviceLoop` allows you to pass in a `LONGINT` of userdata. With this longint, you can pass in the pointer to the rectangle from the assembly routine.

[Back to top](#)

Resizing a Macintosh edit field for TEPaste

Date Written: 11/27/91

Last reviewed: 6/14/93

I have a small edit field and when I call `TEPaste`, the edit field contents scroll up. How can I calculate what the height of the edit field would be after the paste, resize the edit field to be that size, and then do the paste?

—

There are a couple of ways to implement this:

Method #1:

1. Set the clip region to something other than the `viewRect` of the edit field.
2. Call `TEPaste` or `TEStylPaste` to insert the clipboard text.
3. Call `TEGetHeight` to get the new height of the text.
4. Set the `viewRect` of the `TERec` to the new height. Call `TECalText` for good measure (and good `lineStarts`).
5. Finally, set the clip region back to normal and call `TEUpdate`.

Method #2:

1. Call `TENew` or `TEStylNew` to allocate a brand spankin' new temporary `TERec`, and give it a `viewRect` and `destRect` that's off the screen but that has the same width as the original `TERec`.
2. Next, copy the `hText` handle of the window's `TERec` into the `hText` field of the temporary `TERec` (good job for `BlockMove` [IM II-44]).
3. Call `TECalText` and `TEActivate` on it; then paste the clipboard text into the temporary `TERec`. Since the view and dest Rects are off the screen, none of this will be seen by the user.
4. Call `TEGetHeight` to get the new height, then copy the text back into the original `TERec`.
5. Finally, call `TECalText`, then `TEUpdate` and `TEActivate` on the original `TERec`.

Method #1 has less overhead and is a little easier to implement. However, if you have any custom hooks installed in your `TERec`, then you probably want to let TextEdit do as much of the work as possible. In this case, method #2 is a better way.

[Back to top](#)

Saving and restoring styled Macintosh TextEdit records

Date Written: 11/26/90

Last reviewed: 6/14/93

How do I save and restore Macintosh records created with TextEdit that contain styles? Where can I find additional information or sample code?

—

There's no standard for saving text with styles created by TextEdit, so all we can offer is a recommendation that's modeled after the way that `TECopy` works on styled text. In styled TextEdit, the style information is stored separately from the text rather than embedded within it. `TECopy` maintains this separation when styled text is copied, so our recommendation maintains this separation, too.

To save the text part, you could make a file of type `TEXT` and save the text in the data fork of the file, like TeachText does. You could also save it in a resource of type `TEXT`.

The best way to save the style information is to use the `StScrpRec` record. Other records used by styled TextEdit contain references to other structures or they contain only a part of the style information, and that makes saving them awkward. The `StScrpRec` doesn't contain references to any other structures, and it contains enough information to rebuild the entire tree of TextEdit structures. When you call `TECopy` on styled text, both a `'TEXT'` and `'styl'` scrap type is placed into the desk scrap. The `'TEXT'` type just contains the text. The `'styl'` type contains a `StScrpRec` containing all of the style information for the copied text. TextEdit doesn't maintain a `StScrpRec` for the entire text buffer, so you have to

tell TextEdit to do this for you when you're ready to save the style information.

Fortunately, one call does it all. *Inside Macintosh* Volume V, page 268, documents the `GetStylScrap` routine. It creates a new `StScrpRec` record, places all the style information of the selected text into it, and returns a handle to this `StScrpRec`. First, select the text you want to save either by calling `TESetSelect` or by setting the `selStart` and `selEnd` fields of the TextEdit record if you don't want the selection to be visible to the user. (If you do this, remember to restore the old values of `selStart` and `selEnd` after you get the `StScrpHandle`.) Next, call `GetStylScrap` to construct the `StScrpRec`. You can now save the resulting `StScrpRec`, perhaps in a resource of type 'styl' or in whatever way you see fit.

Reading the text and styles back into a TextEdit record isn't much more complicated. First, read the text back into a buffer and read the 'styl' resource (or however you saved it) into a `StScrpRec`. Create a styled TextEdit record (or use an old one), then call `TEStylInsert`, passing it a pointer to the text buffer and a handle to the `StScrpRec`. Your TextEdit record should then contain and display your original styled text. That's all!

[Back to top](#)

Macintosh SetClikLoop procedure and Terec clikLoop field

Date Written: 7/30/91

Last reviewed: 6/14/93

I have had some problems with the `SetClikLoop` procedure that don't appear when I set the `clikLoop` field directly (in C). Why does TextEdit's `SetClikLoop` procedure create a special procedure to call my `clikLoop` routine, rather than just installing the `clikLoop` directly? *Inside Macintosh* says you can set the `clikLoop` field directly in assembly. Why not in C?

—

The short answer is that the special procedure is a glue routine for shuttling a result between the stack and the register D0. The larger question is why does installing `clikLoop` directly work? C, I think, is the culprit.

But, before I get ahead of myself, let me explain the `SetClikLoop` procedure. As you know, the click loop routine has no parameters but it does return a Boolean result (IM I-380).

```
FUNCTION MyClikLoop : Boolean; { Pascal declaration }
```

Now, `MyClikLoop` is called by the TextEdit Manager from inside the `TEClick` routine. The way the code was written, `TEClick` expects to get the Boolean result from D0. For assembly programmers, this is no problem. In Pascal, the return value is placed on the stack. So, D0 does not contain your routines' Boolean and what `TEClick` grabs will be "who knows what" and your program should crash. `SetClikLoop` will install a glue routine that calls `MyClikLoop`. The glue routine will move the result from the stack to D0. By the way, this information is documented on pages 237-238 of Volume Two of *Macintosh Revealed*, Second Edition, by Stephen Chernicoff.

Now why is your program causing problems when you use `SetClikLoop`? You might have declared your `MyClikLoop` as a C function. Since `MyClikLoop` takes no parameters, nothing is put on the stack. But C, unlike Pascal, places its return value in D0. So, for C, the glue routine is not needed. On the other hand, if you use the `SetClikLoop` routine with a C routine, you will take the top 2 bytes off the stack and clobber D0 with the wrong answer. Thus, you might get problems.

Though the C routine may work by setting it directly to the `clikLoop` field, doing it that way is not the supported method. You should declare your `MyClikLoop` function as a Pascal function and use `SetClikLoop`.

[Back to top](#)

Special handling for Control-P keyboard input in a Mac dialog

Date Written: 6/5/91

Last reviewed: 6/14/93

Under System 6, when Control-P is pressed, a DLE character (hex 10) appears in a Macintosh dialog's editText field. Under System 7, pressing Control-P produces no change in the editText box. The character is fed in via a normal (unaltered) `keyDown` event fed to `DialogSelect`. Does Control-P have some special meaning to `DialogSelect` or TextEdit now?

Believe it or not, and it took several of us scratching our heads to figure this out; yes, it does, and this is by design! What you are running into is the Dialog Manager and TextEdit working together to make Cut, Copy, and Paste function keys work on an extended keyboard. The function keys on the top of an extended keyboard return \$10 as their ASCII keycode. Thus, when `TEKey` detects that it is being called from dialog select, and the key you pass is a \$10, it then looks for the event record on the stack and gets the raw keycode to determine which edit operation to perform. The bottom line is that it will not then enter the \$10 in the edit field.

There is a simple workaround, and that is to install a filter proc in your dialog that detects keystrokes (actually it could always just look for this certain keystroke) and passes them to `TEKey` itself, thus skirting the whole issue.

[Back to top](#)

Dimming and disabling a specific TEditText object

Date Written: 8/8/91

Last reviewed: 8/13/91

How can I dim and disable a specific TEditText object in my TDialogView so the field is not editable?

The trick is to use the `TView` methods `DimState` and `ViewEnable` to disable and dim the `TEditText` Views, and then do a `TEditText.StopEdit` (which disables editing in the `TEditText` box), and in the other case enable dimming and the view itself and use the `TEditText.StartEdit` (which enables editing in the `TEditText` box), as shown in the C++ example below:

```
anEditText2->DimState(TRUE,kDontRedraw);           // disable
anEditText2->ViewEnable(FALSE,kRedraw);
anEditText2->StopEdit();

anEditText1->DimState(FALSE,kDontRedraw);           // enable
anEditText1->ViewEnable(TRUE,kRedraw);
```

MacApp sample code using this technique is included with DTS's Snippets files.

[Back to top](#)

Get TEHandle from DialogPeek, not GetDItem

Date Written: 1/17/92

Last reviewed: 6/14/93

I want to show all characters in my Macintosh control panel as bullets where the password is typed in. I cannot get the `TECustomHook` to work properly, it crashes before it gets to my hook routine. In the following Think C example, `CPtr` is dereferenced from the handle used in the control panel's `cdevValue` field:

```
GetDItem(DPptr, ADMINPWTE+NumItems, &theType, &theItem, &theRect);
CPtr->myTEProc = (ProcPtr)TEPWHook;
```

`GetDItem` does NOT return a `TEHandle`; it returns a handle to the current text of the edit item. You must get `TEHandle` from `DialogPeek` and must always monitor and change the `TECustomHook` on the fly. There is only one `TEHandle` for a dialog, no matter how many edit items are associated with the dialog. See the description of the `DialogRecord` data type in *Inside Macintosh*.

[Back to top](#)

Cursor flicker while using Macintosh TextEdit

Date Written: 5/6/92

Last reviewed: 6/14/93

The cursor flashes when the user types in TextEdit fields in my Macintosh application. This is done in `TEKey`. I notice that most programs hide the cursor once a key is pressed. I don't care for this because then I have to move the mouse to see where I am. Is this a typical fix for this problem and an accepted practice?

There's very little you can do to avoid this. The problem is that every time you draw anything to the screen, if the cursor's position intersects the rectangle of the drawing being done, QuickDraw hides the cursor while it does the drawing, and then shows it again to keep it from affecting the image being drawn beneath it. Every time you enter a character in TextEdit, the nearby characters are redrawn. Usually this is invisible because the characters just line up on top of their old images, but if the cursor is nearby and visible, it will flicker while it's hidden to draw the text. This is why virtually all programs call `ObscureCursor` when the user types. Also, most users don't want the image of the cursor obscuring text they might be referring to, yet they don't want to have to move it away and then move it back to make selections. Because it's so commonplace, hiding the cursor probably won't bother your users; in fact, they might very well prefer the cursor hidden. This, combined with the fact that there's very little you can do to help the flickering, suggests that you should obscure the cursor while the user types.

[Back to top](#)

Downloadables



Acrobat version of this Note (K).

[Download](#)

Technical Notes by [Date](#) | [Number](#) | [Technology](#) | [Title](#)
[Developer Documentation](#) | [Technical Q&As](#) | [Development Kits](#) | [Sample Code](#)