When will the static control automatically delete the image loaded into it, and when is it the responsibility of the application?

devblogs.microsoft.com/oldnewthing/20140219-00

February 19, 2014

Raymond Chen

If you create a static control with initial contents (for example, by creating a `BITMAP` or `ICON` control in a dialog template), then the static control will load the contents upon creation and destroy the contents upon destruction. So at least in the case where you don't touch the static control, things will work automatically.

But once you touch it, things get confusing.

If you send the `STM_SETIMAGE` message to a static control, this does a few things (assuming your parameters are all valid):

- The previous image is replaced by the new image you passed.
- The message returns a handle to the previous image.
- The static control *turns off automatic image deletion*.

The third part is the tricky part. If you ever (successfully) send a static control the `STM_SETIMAGE` message, then it says, "Okay, it's all your problem now." You are now responsible not only for destroying the new image, but you are also responsible for destroying the old image that was returned.

In other words, the following operation is *not* a nop:

```
HBITMAP hbmPrev = SendMessage(hwndStatic, STM_SETIMAGE,
                              IMAGE_BITMAP, (LPARAM)hbmNew);
SendMessage(hwndStatic, STM_SETIMAGE,
            IMAGE_BITMAP, (LPARAM)hbmPrev);
```

This sounds like a nop, since all you did was change the image, and then change it back. But the side effect is also that you made the static control go into *your problem* mode, and the original image will no longer be automatically destroyed. If you forget to destroy it yourself, then you have a leak.

Wait, it gets worse.

If you are using version 6 of the common controls, then things get even more confusing if you use the `STM_SETIMAGE` message to change the `IMAGE_BITMAP` of a `SS_BITMAP` static control, and the bitmap you pass is a 32-bpp bitmap, and the image has a nonzero alpha channel, then the static control will make a *copy* of the bitmap you passed in and act as if you had passed that copy instead.[1] This by itself is no big deal, because the responsibility for destroying the image you passed in still resides with you, the application, so the rules haven't changed there.

The nasty bit is that the application also must assume responsibility for *destroying the secret copy*. That bitmap you didn't even know existed and don't have a handle to? Yeah, you're on the hook for that one too.

How unfair.

Even more confusing is that if you send `STM_SETIMAGE` a second time, it will replace the bitmap and return a handle to *the secret copy* (which is a bitmap you've never seen before).

This means that the following assertion can fire:

```
HBITMAP hbmPrev = SendMessage(hwndStatic, STM_SETIMAGE,
                              IMAGE_BITMAP, (LPARAM)hbmNew);
HBITMAP hbmBack = SendMessage(hwndStatic, STM_SETIMAGE,
                              IMAGE_BITMAP, (LPARAM)hbmPrev);
assert(hbmNew == hbmBack); // ??
```

You would think that the assertion is safe because all you did was change the bitmap to `hbmNew`, then change it back. And when you change it back, the "previous value" is the value `hbmNew` you set it to on the previous line.

Except that if `hbmNew` satisfies the above magic criteria, then the value in `hbmBack` is not `hbmNew` but rather the handle to the *secret copy*.

Which you have to remember to destroy.

Yuck.

The secret copy is not *too* secret. You can get a handle to it by sending the `STM_GETIMAGE` message. Which you now need to do when you destroy the static control, just in case it's the *secret copy*. You need to compare the current image against the one that you thought you passed in, and if they are different, then you have the *secret copy* that needs to be destroyed as an extra step.

Yes, this sucks. I apologize.

(My recommendation: To detect whether a “secret copy” occurred, do a `STM_GETIMAGE` after your `STM_SETIMAGE` and see if the handles match.)

[1] The *secret copy* is not an exact copy. (After all, if it were an exact copy, then there would be no need to create the copy. It could just use the handle you passed in.) Instead, the *secret copy* is a copy of the original, followed by some additional munging so that it can be displayed on the screen while respecting the alpha channel you passed in.

Raymond Chen

Follow