Does version 6 of the common controls support ANSI or not?

devblogs.microsoft.com/oldnewthing/20081106-00

November 6, 2008

Raymond Chen

I mentioned in passing a few years ago that version 6 of the common controls supports only Unicode. And then other people stepped in to say, “Well, XYZ uses ANSI and that works for me.” So does it support ANSI or doesn’t it?

It does and doesn’t.

All of the controls in the common controls library are internally Unicode. But not all controls in the library are created equal.

The first group is the traditional common controls. List view, tree view, those guys. These controls were never part of the window manager and have been internally Unicode on all Windows NT platforms. The ANSI messages such as `LVM_SETITEMA` are implemented by thunking to and from Unicode.

The second group is the controls that were traditionally part of the window manager itself. If you aren’t using version 6 of the common controls, you will continue to use the versions built into the window manager, and those versions, for the most part, are also internally Unicode.

The one weirdo is the edit control. The edit control uses black magic voodoo to tell whether you created it with `CreateWindowExA` or `CreateWindowExW`, and its internal edit buffer is ANSI or Unicode accordingly. (Regular window classes don’t have access to this magic voodoo. It’s one of the historical weirdnesses of the edit control that date back to the old days.)

The internal character set goes largely unnoticed since the window manager automatically converts between Unicode and ANSI as necessary. For example, if you call `SetWindowTextA` to a Unicode edit control, the window manager will convert the string from ANSI to Unicode and send the Unicode string to the edit control. The one place the internal character set becomes visible to the outside world is with the `EM_GETHANDLE` and

`EM_SETHANDLE` messages, because these messages access the internal buffer of the edit control. You therefore have to know whether your edit control is a Unicode or ANSI edit control so you know the correct format of that internal buffer.

When these window manager controls were ported into the common controls library, the voodoo was lost, since that magic is available only to internal window manager classes, and the common controls aren't internal window manager classes. Since the common controls library uses `RegisterClassW` to register the window class, the edit control that comes with the common controls is a Unicode edit control. In other words, if you use `CreateWindowA` to create an edit control from the common controls library, and you send it a `EM_GETHANDLE` message, the buffer you get back will be a *Unicode* buffer, not an ANSI one.

This wacky behavior with `EM_GETHANDLE`, as well as other even more subtle weirdnesses that come from the edit control in the common controls library being always internally Unicode means that code that calls `CreateWindowA` and expects the result to be an edit control which is internally ANSI will be in for a bit of a surprise when they are using version 6 of the common controls library.

These and other subtle ANSI/Unicode discrepancies are why the common controls library, starting with version 6, requires a Unicode application. If you're an ANSI application and you create controls from the common controls library, you may encounter strange behavior. It'll mostly work, but things may be weird at the fringe.

Now, why not just get rid of all the ANSI support entirely? Why leave it in, even though it doesn't quite work perfectly? For the same reason the Windows XP common controls are not a separate library with separate window class names. As noted, there are programs that like to go hunting around into windows that don't belong to them. Some of those programs might stumble upon one of Explorer's list views and use various nefarious techniques to do things like stealing strings from another program's list view control. If support for the ANSI messages such as `LVM_GETITEMA` were removed entirely, then those sneaky programs would stop working.

You might say, "Well, tough for them." You'll say that until you discover that one of those sneaky programs happens to be one that you use every day, possibly even one that you wrote yourself. Oops. Now you're going to tell all your friends, "Don't upgrade to the next version of Windows. Its compatibility sucks."

Okay, so the common controls still have to be backward compatible with the ANSI messages that existed in version 5. But at least the new messages such as `LVM_SETINFOTIP` can be Unicode-only.

And it means that all you folks who are using version 6 of the common controls but haven't converted to Unicode are relying on a compatibility loophole. The ANSI support is there for the old programs that thought they were talking to a version 5 common control; it isn't there for you.

Raymond Chen

Follow