The history of the Windows XP common controls

devblogs.microsoft.com/oldnewthing/20080129-00

January 29, 2008

Raymond Chen

In the beginning, there was one control library, namely `USER`, the window manager itself, which provided buttons, static controls, edit controls, scroll bars, list boxes, and combo boxes. These controls were under the purview of the window manager team.

In Windows 3.1 a second control library known as the shell common controls was added, but the library really didn't come into it own until Windows 95, where it consisted of the list view, header, tree view, tooltip, toolbar, status bar, track bar, tab, updown, progress, hotkey, and animation controls. These controls were originally custom controls written by the shell team for use in Explorer, but since they seemed to be generally useful, time was budgeted to do the extra work to make the controls more suitable for general use, testing the controls in combinations and scenarios that Explorer itself didn't use, and putting together formal documentation.

The shell common controls library underwent many changes over the years, whereas the core intrinsic controls in the window manager changed much more conservatively.

With Windows XP, the visual design team wanted to give the look of Windows a new life. Changing the non-client area (such as the window frame) was comparatively straightforward, since programs didn't have much control over that part of the window anyway. As a result, they get the Windows XP look "for free".

The client area is another story. Programs are in control of their client area, where they can place controls that come with Windows, their own custom controls, or controls obtained from a third party library. Making major changes to the core controls or the common controls would be a high-risk endeavor since there are thousands upon thousands of Windows program that not only use them, but use them in all sorts of crazy ways.

The initial stab at resolving these two conflicting goals (making major changes to these controls to increase visual appeal and functionality while simultaneously not changing them to maintain compatibility) was to create a new DLL that would contain the "fancy new version" of the core controls as well as the shell common controls. The old DLLs `USER32` and `COMCTL32` stayed where they were, so that old programs continued to get the behavior

they were expecting, and the new XP-style controls were placed in a DLL named `UXCTRL.DLL`. UX stands for *user experience*, which was the hot new buzzword at the time. To avoid name collision with the old style controls, the new controls got new names beginning with Ux. For example, the `UXCTRL` version of the button control was called UxButton.

New features could be added to these new Ux controls with wild abandon without heed for backward compatibility since they were brand new controls. There was nothing they had to be compatible with. Explorer was changed to use these new controls instead of the old stodgy controls, and everything worked great.

Or so it seemed.

We thought we had cleverly sidestepped the backward compatibility problem by creating entirely new controls, but doing that created *a whole new category* of compatibility bugs. Even though it's completely undocumented and unsupported, programs like to grovel into the internal data structures of other programs or otherwise manipulate those programs' windows. In Explorer's case, it turns out that a lot of programs like to go spelunking around Explorer's window hierarchy and use functions like `FindWindow` and `EnumChildWindows` to find the object of their affections. For example, a program might use `EnumChildWindows` to enumerate all the child windows of an Explorer browser, and then use `GetClassName` and `lstrcmpi(szClassName, TEXT("button"))` to look for a specific control. In this example, the target was a button, but it could have been a list view or a tool bar. Since all the new XP-style controls were named things like UxButton and UxListView, these programs which looked for a button by comparing against the string `"button"` stopped working.

Of course, there was no guarantee that Explorer would even use buttons at all; Explorer was completely within its rights to revamp its user interface. But that's not much consolation to the customers who paid good money for these programs, especially since magazine columnists are the types of people most likely to be running (or indeed even writing!) strange off-the-wall programs that pull these sorts of nasty stunts in the first place.

Okay, so it is now a compatibility requirement that all the new window classes have the same names as their old counterparts. This created an impasse, since these controls needed to be created by dialog boxes, and therefore they had to be globally-registered window classes. But you can't have two global window classes with the same name, because that would create ambiguity over which one the caller was asking for.

More brainstorming ensued, and a Plan C emerged. The common controls library would take advantage of side-by-side assemblies and use the application manifest to control which DLL a given window class name would resolve to. Thus was born a new DLL also called

`COMCTL32`, but with a new version number—version 6. Old programs would get version 5.82 just like they did in Windows 2000. New programs would have to use a manifest to specify that they wanted version 6.

Once again, the solution came with a new problem. Since the entire `COMCTL32` library got split into two versions, this meant that there were two versions of the image list code. Whole new scenarios emerged, such as putting a version 5 image list in a version 6 tree view, or vice versa. (As the linked thread notes illustrates, not all of the problems with cross-version scenarios were caught in the initial release and had to wait for a service pack for the fix to become available.)

Raymond Chen

Follow