What is the default version of the shell common controls?

devblogs.microsoft.com/oldnewthing/20070412-00

April 12, 2007

Raymond Chen

It depends on what you mean by default. As we saw earlier, the convention for Windows header files is that if you don't specify a particular version, then you get the most recent version. The shell common controls header file follows this convention, so if you include the Windows XP version of `commctrl.h`, you get functions, messages, and structures designed for use with version 6 of the common controls. (And functions, messages, and structures may not work with version 5 of the shell common controls due to changes in structure sizes, for example.) So from the Windows XP Platform SDK header file's point of view, the default version of the shell common controls is version 6. On the other hand, there's the question of what version of the shell common controls you actually get at run time. Prior to Windows XP, the answer was simple: You got the most recent version installed on the machine. With Windows XP, however, the rules changed. The visuals team wanted to do something more ambitious with the common controls, but the compatibility constraints also created significant risk. The solution was to use side-by-side assemblies. For compatibility, if a program didn't specify what version of the shell common controls it wanted, it got version 5.82, which was carefully designed for extremely high compatibility with the previous version, 5.81, which came with Windows 2000 and Windows Me. Now, version 5.82 is not completely identical to 5.81, because it also needs to interoperate with version 6. More on this later. If a program wanted to use version 6 of the common controls, it had to say so explicitly in a manifest. (What we on the shell team informally call a "v6 manifest".) That way, only programs that asked for the new behavior got it. The theory being that if you asked for the new behavior, you presumably tested your program against version 6 of the common controls to verify that it behaves as you expected. This freed up the visuals team to make more substantial changes to the common controls without having to worry about some old program that relied on some strange undocumented behavior of the common controls. That old program would get version 5.82, which was designed for high compatibility. Now, on that interoperability thing. There are places where the common controls library creates an object which you can then use with other common controls. For example, you can create an image list with `ImageList_Create` and then use that image list in a list view or tree view. Care had to be taken so that an image list created by version 5 of the common controls (a "v5 image list") could be used by a list view created by version 6 (a "v6 list view"), or conversely that a v6 image list could be used in a v5 list view. This sort of cross-version image list usage is actually quite common: Any application that calls `Shell_GetImageLists` (or its old-fashioned equivalent, `SHGetFileInfo` with the `SHGFI_SYSICONINDEX` flag) will get a v6 image list. If that application uses version 5 of the common controls (because it doesn't have a v6 manifest), then it will find itself using a v6 image list inside a v5 list view. Since each DLL has its own manifest, you can quickly find yourself in a case where there is a hodgepodge of v5 and v6 components all inside a single process, and they all have to work with each other. Another example of this cross-version interoperability is the `HPROPSHEETPAGE`. Property sheet pages created with `CreatePropSheetPage` from one version of the shell common controls had to work with the `PropertySheet` function of the other version. This happens a lot with shell property sheet extensions. The shell namespace will ask the shell extensions to provide their custom property sheets, and all the ones written for Windows 2000 will hand back a v5 `HPROPSHEETPAGE`. But Explorer is going to display that property sheet with the v6 `PropertySheet` function. That v5 property sheet page had better work even when hosted inside a v6 property sheet.

Okay, but back to the original problem. If you don't specify what version of the header file you want, then you get the latest version (version 6 if you got the header file from the Windows XP Platform SDK). On the other hand, if you don't specify what version of the DLL you want, you get version 5.82, the compatible version of the DLL. *Yes, this is a mismatch. Be on the lookout*. This is what happens when a header file convention is at odds with a compatibility decision.

Raymond Chen

Follow