Why doesn't Win32 give you the option of ignoring failures in DLL import resolution?

devblogs.microsoft.com/oldnewthing/20100913-00

September 13, 2010

Raymond Chen

Yuhong Bao asked, via the Suggestion Box, "Why not implement delay-loading by having a flag in the import entry specifying that Windows should mimic the Windows 3.1 behavior for resolving that import?" Okay, first we have to clear up the false assumptions in the question. The question assumes that Windows 3.1 had delay-loading functionality in the first place (functionality that Yuhong Bao would like added to Win32). Actually, Windows 3.1 behavior did not have any delay-load functionality. If your module imported from another DLL in its import table, the target DLL was loaded when your module was loaded. There was no delay. The target DLL loaded at the same time your module did. So there is no Windows 3.1 delay-load behavior to mimic in the first place. Okay, maybe the question really was, "Instead of failing to load the module, why not just let the module load, but set the imported function pointers to a stub function that raises an error if you try to call it, just like Windows 3.1 did?" Because it turns out that the Windows 3.1 behavior resulted in data loss and mystery crashes. The Win32 design solved this problem by making failed imports fatal up front (a design principle known as *fail fast*), so you knew ahead of time that your program was not going to work rather than letting you run along and then watch it stop working at the worst possible time, and probably in a situation where the root cause is much harder to identify. (Mind you, it may stop working at the worst possible time for reasons the loader could not predict, but at least it stopped what it could.) In other words, this was a situation the Win32 people thought about and made an explicit design decision that this is a situation they would actively not support. Okay, but when Visual Studio was looking at how to add delay-load functionality, why didn't they implement it by changing the Win32 loader so that failed imports could be optionally marked as non-fatal? Well, um, because the Visual Studio team doesn't work on Windows? There's this feature you want to add. You can either add it to the linker so that all programs can take advantage of the feature on all versions of Windows, or you can add it to the operating system kernel, so that it works only on newer versions of Windows. If the feature had been added to the loader rather than the linker, application vendors would say, "Stupid Microsoft. I can't take advantage of this new feature because a large percentage of my customer base is still running the older operating system. Why couldn't they have added this feature to the linker, so it would work on all operating systems?" (You hear this complaint a lot. Any time a new version of Windows adds a feature, everybody demands that it be ported

downlevel.) Another way of looking at this is realizing that you're adding a feature to the operating system which applications can already do for themselves. Suppose you say, "Okay, when you call a function whose import could not be resolved, we will display a fatal application error." The response is going to be "But I don't want my application to display a fatal application error. I want you to call this error handler function instead, and the error handler will decide what to do about the error." Great, now you have to design an extensibility mechanism. And what if two DLLs each try to install different failed-imported-function handlers?

When you start at minus 100 points, saying, "Oh, this is not essential functionality. Applications can simulate it on their own just as easily, and with greater flexibility" does nothing to get you out of the hole. If anything, it digs you deeper into it.

Raymond Chen

Follow